Analysis of Cross-Sectional Data

Kevin Sheppard

Course Structure

- Course presented through three channels:
    1. Pre-recorded content with a focus on technical aspects of the course
        - Designed to be viewed in sequence
        - Each module should be short
        - Approximately 2 hours of content per week
    2. In-person lectures with a focus on applied aspects of the course
        - Expected that pre-recorded content has been viewed before the lecture
    3. Notes that accompany the lecture content
        - Read before or after the lecture or when necessary for additional background
- Slides are primary – material presented during lecturers, either pre-recorded or live is examinable
- Notes are secondary and provide more background for the slides
- Slides are derived from notes so there is a strong correspondence

Monitoring Your Progress

- Self assessment
    - Review questions in pre-recorded content
    - Multiple choice questions on Canvas made available each week
        - Answers available immediately
    - Long-form problem distributed each week
        - Answers presented in a subsequent class
- Marked Assessment
    - Empirical projects applying the material in the lectures
    - Both individual and group
    - Each empirical assignment will have a written and code component

Analysis of Cross-Sectional Data

Introduction to Regression Models

- Notation
- Factor Models
- Data
- Variable Transformations

Linear Regression

Scalar Notation

$$Y_i = \beta_1 X_{1,i} + \beta_2 X_{2,i} + \ldots + \beta_k X_{k,i} + \epsilon_i$$

- Y_i: Regressand, Dependent Variable, LHS Variable
- $X_{j,i}$: Regressor, also Independent Variable, RHS Variable, Explanatory Variable
- ϵ_i: Innovation, also Shock, Error or Disturbance
- n observations, indexed $i = 1, 2, \ldots, n$
- k regressors, indexed $j = 1, 2, \ldots, k$

Linear Regression

Matrix Notation

Common to use convenient matrix notation

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

- $\mathbf{y}$ is n by 1
- $\mathbf{X}$ is n by k
- $\boldsymbol{\beta}$ is k by 1
- $\boldsymbol{\epsilon}$ is n by 1

Factor Models

- Factor models are widely used in finance
  - Capital Asset Pricing Model (CAPM)
  - Arbitrage Pricing (APT)
  - Risk Exposure
- Basic specification $R_i = \mathbf{f}_i\boldsymbol{\beta} + \epsilon_i$
  - R_i: Return on dependent asset, often excess (R_i^e)
  - $\mathbf{f}_i$: $1 \times k$ vector of factor innovations
  - ϵ_i innovation, $corr(\epsilon_i, F_{j,i}) = 0$, $j = 1, 2, \ldots, k$
- Special Case: CAPM

$$R_i - R_i^f = \beta(R_i^m - R_i^f) + \epsilon_i$$

$$R_i^e = \beta R_i^{me} + \epsilon_i$$

Data

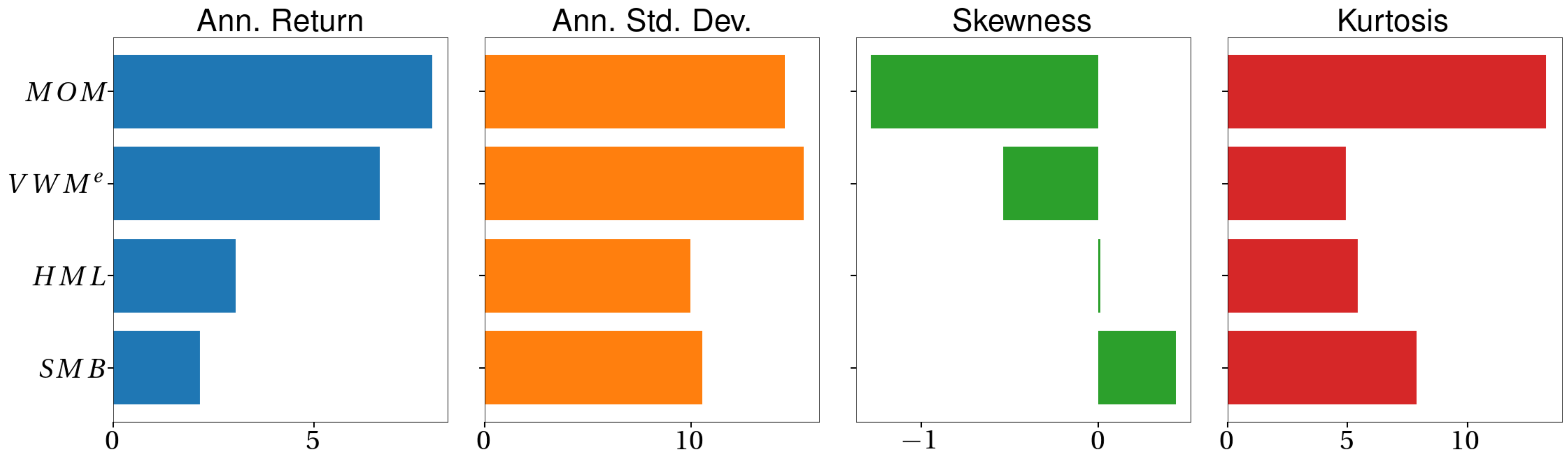
- Data from the Fama-French 3 factors + Momentum
  - VWM^e - Excess return on Value-Weighted-Market
  - SMB - Return on the size portfolio
  - HML - Return on the value portfolio
  - MOM - Return on the momentum portfolio
- Size-Value sorted portfolio return data
  - Size
    - S: Small
    - B: Big
  - Value
    - H: High BE/ME
    - M: Middle BE/ME
    - L: Low BE/ME
- 49 Industry Portfolios
- All returns excess except SMB, HML, MOM

Fama-French Factors

Summary Statistics

In [3]:

```
summ_plot(factors)
```



Fama-French Factors

Correlation Structure

In [4]:

```
factors.corr()
```

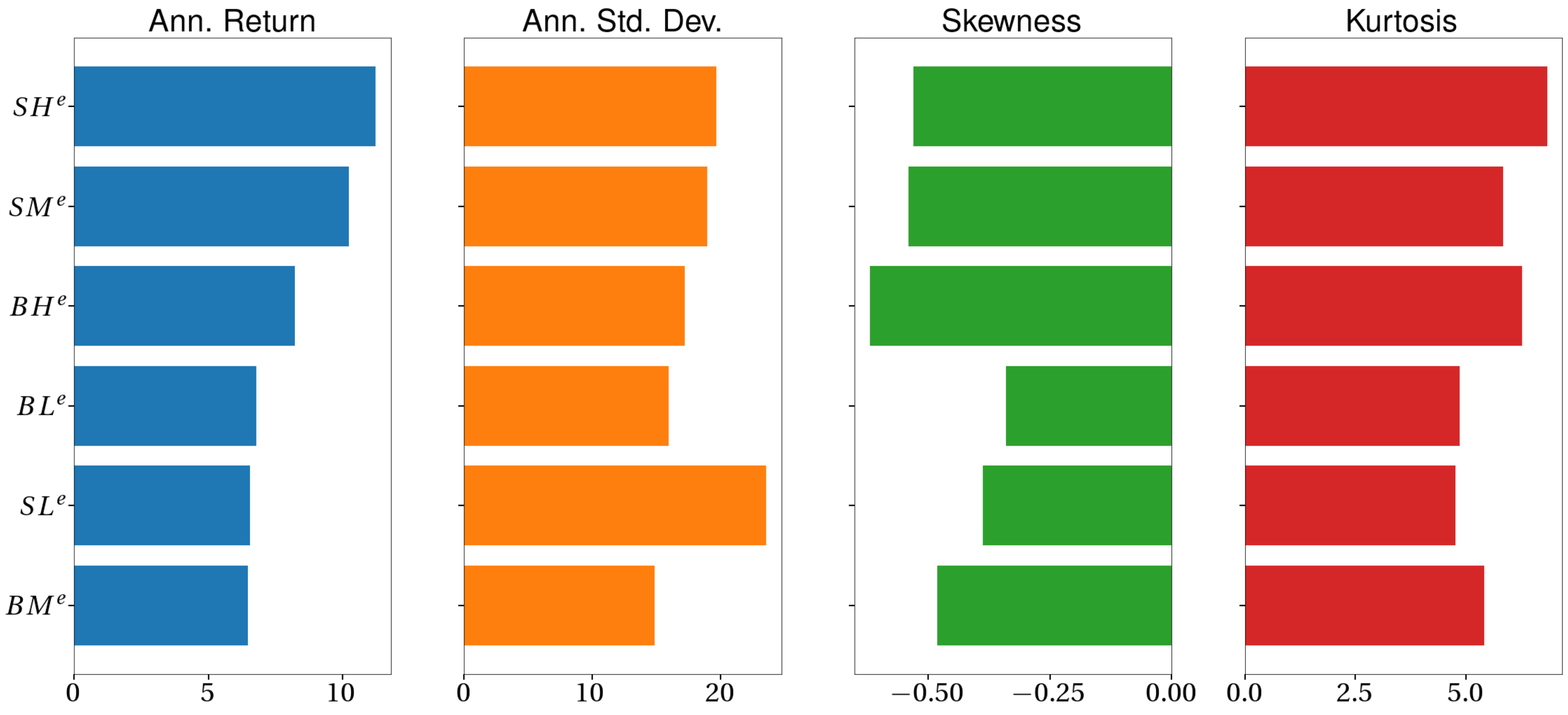
Out[4]:

	VWM^e	SMB	HML	MOM
VWM^e	1.000000	0.300958	-0.226222	-0.149518
SMB	0.300958	1.000000	-0.174962	-0.024014
HML	-0.226222	-0.174962	1.000000	-0.195242
MOM	-0.149518	-0.024014	-0.195242	1.000000

Size and Value components

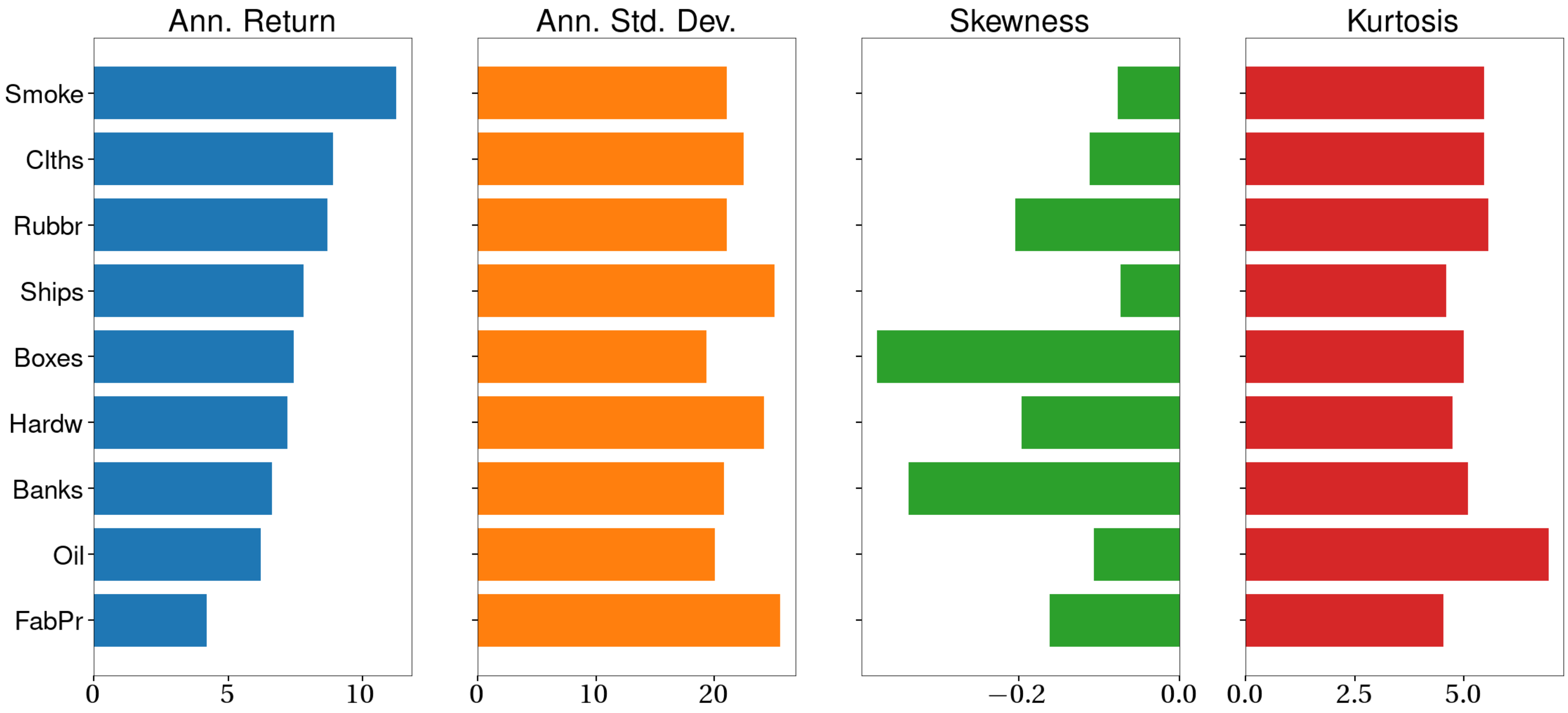
In [6]:

```
summ_plot(components)
```



Industry Portfolios

In [7]:
```
summ_plot(subset)
```



Variable Transformations

- Dummy variables
    - 0-1 variables based on an indicator function

$$I_{[X_{i,j}>0]}$$

- Asymmetries at 0
- Monthly Effects

In [9]:

```
monthly_dummies.head(8)
```

Out[9]:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	0	0	0	0	1
6	1	0	0	0	0	0	0	0	0	0	0	0
7	0	1	0	0	0	0	0	0	0	0	0	0

Variable Transformation: Interactions

- Interactions dramatically expand the functional forms that can be specified
  - Powers and Cross-products: $X_{i,j}^2$, $X_{i,j}X_{i,m}$
  - Dummy Interactions to Produce Asymmetries: $X_{i,j} \times I_{[X_{i,j}<0]}$

In [11]:

```
interactions.tail(10)
```

Out[11]:

	Market Negative	Negative Return	Squared Returns
2019-11-30	0	0.00	14.9769
2019-12-31	0	0.00	7.6729
2020-01-31	1	-0.11	0.0121
2020-02-29	1	-8.13	66.0969
2020-03-31	1	-13.38	179.0244
2020-04-30	0	0.00	186.3225
2020-05-31	0	0.00	31.1364
2020-06-30	0	0.00	6.0516
2020-07-31	0	0.00	33.2929
2020-08-31	0	0.00	58.0644

Analysis of Cross-Sectional Data

Parameter Estimation and Model Fit

- Parameter Estimation
- Models with Interactions
- Other estimated quantities
- Regression Coefficient in Factor Models

Parameter Estimation

Least Squares

$$\text{argmin}_{\beta} \sum_{i=1}^{n} (Y_i - \mathbf{x}_i \boldsymbol{\beta})^2$$

In [13]:

```
ls = smf.ols("BHe ~ 1 + VWMe + SMB + HML + MOM", data).fit(cov_type="HC0")
summary(ls)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.0859	0.043	-1.991	0.046	-0.170	-0.001
VWMe	1.0798	0.012	93.514	0.000	1.057	1.102
SMB	0.0019	0.017	0.110	0.912	-0.032	0.036
HML	0.7643	0.021	36.380	0.000	0.723	0.805
MOM	-0.0354	0.013	-2.631	0.009	-0.062	-0.009

Least Absolute Deviations

$$\text{argmin}_{\beta} \sum_{i=1}^{n} |Y_i - \mathbf{x}_i \boldsymbol{\beta}|$$

In [14]:

```
lad = smf.quantreg("BHe ~ 1 + VWMe + SMB + HML + MOM", data).fit(q=0.5)
summary(lad)
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.0306	0.044	-0.696	0.487	-0.117	0.056
VWMe	1.0716	0.010	103.257	0.000	1.051	1.092
SMB	0.0161	0.015	1.090	0.276	-0.013	0.045
HML	0.7503	0.016	47.702	0.000	0.719	0.781
MOM	-0.0272	0.011	-2.581	0.010	-0.048	-0.007

Estimating Models with Interactions

Added an asymmetry and a square of VWM to the 4-factor model

$$Util_i = \beta_1 + \beta_2 VWM_i^e + \beta_3 \left(VWM_i^e\right)^2 + \beta_4 VWM_i^e I_{[VWM_i^e<0]} + \beta_5 SMB_i + \beta_6 HML_i + \beta_7 MOM_i + \epsilon_i$$

In [15]:

```
model = f"Util ~ 1 + VWMe + I(VWMe**2) + I(VWMe * (VWMe < 0)) + SMB + HML + MOM"
ls_interact = smf.ols(model, data).fit(cov_type="HC0")
summary(ls_interact)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	0.2857	0.225	1.268	0.205	-0.156	0.727
VWMe	0.4594	0.089	5.154	0.000	0.285	0.634
I(VWMe ** 2)	0.0159	0.007	2.240	0.025	0.002	0.030
I(VWMe * (VWMe < 0))	0.3524	0.188	1.870	0.061	-0.017	0.722
SMB	-0.1972	0.048	-4.087	0.000	-0.292	-0.103
HML	0.3470	0.060	5.810	0.000	0.230	0.464
MOM	0.0611	0.039	1.578	0.114	-0.015	0.137

Expected and Fitted Values

- Fitted values:

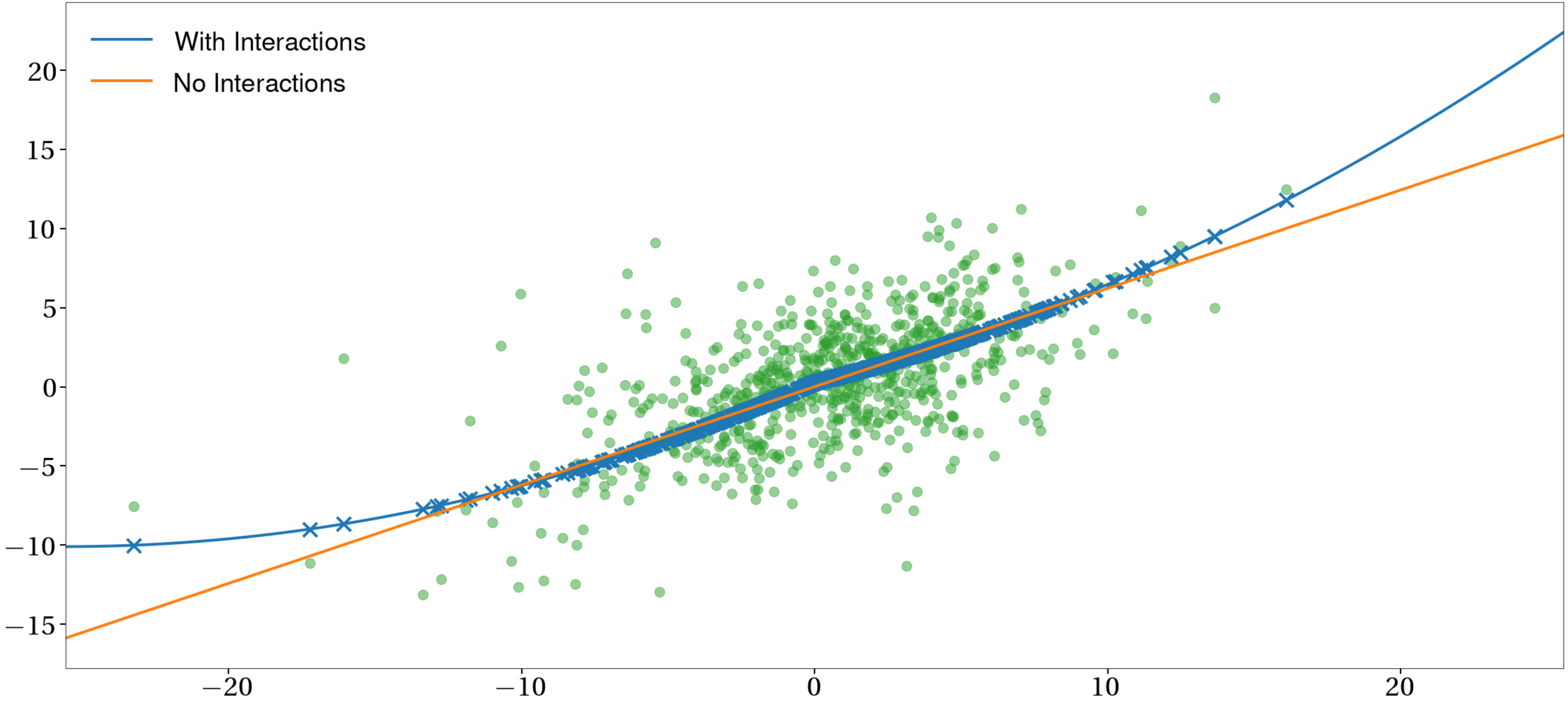
$$\hat{Y}_i = \mathbf{x}_i \hat{\boldsymbol{\beta}}$$

- Expected values:

$$E[Y|X = \mathbf{x}] = \mathbf{x}\hat{\boldsymbol{\beta}}$$

Expected and Fitted Values

In [18]:
```
plot_market_interactions()
```

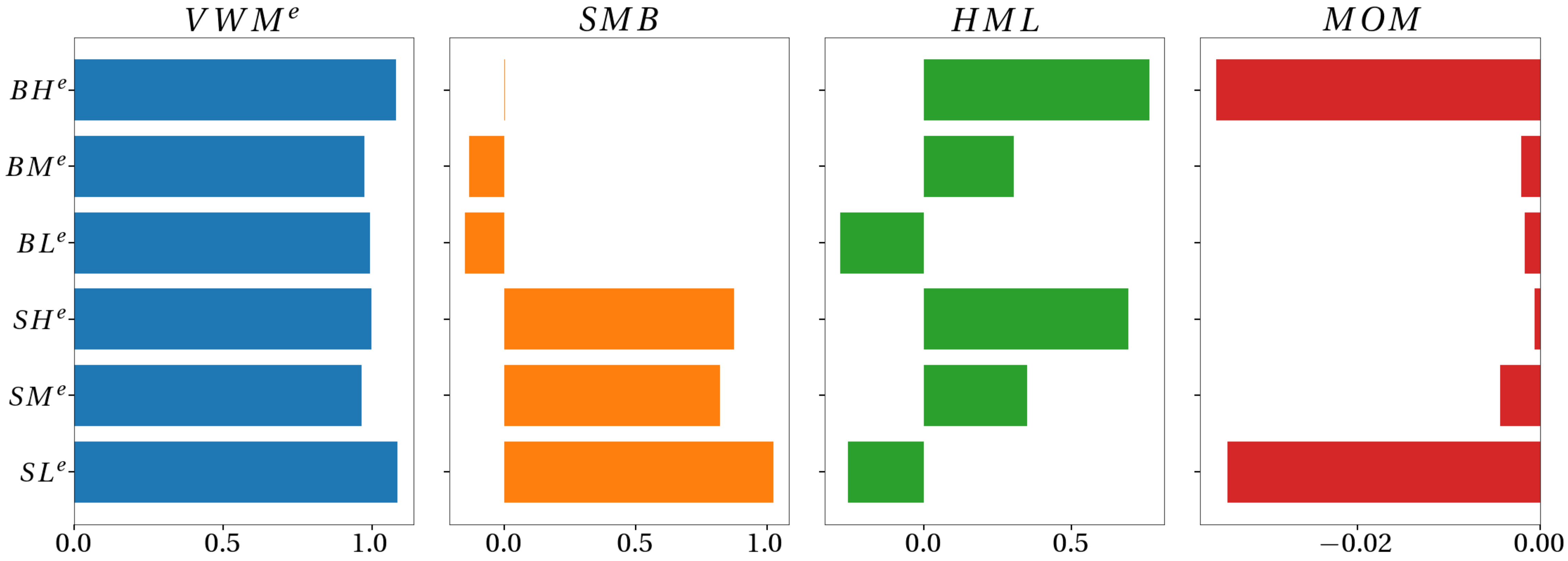


Typical Regression Coefficients

Factor Components

In [20]:

```
beta_plot(betas, titles)
```

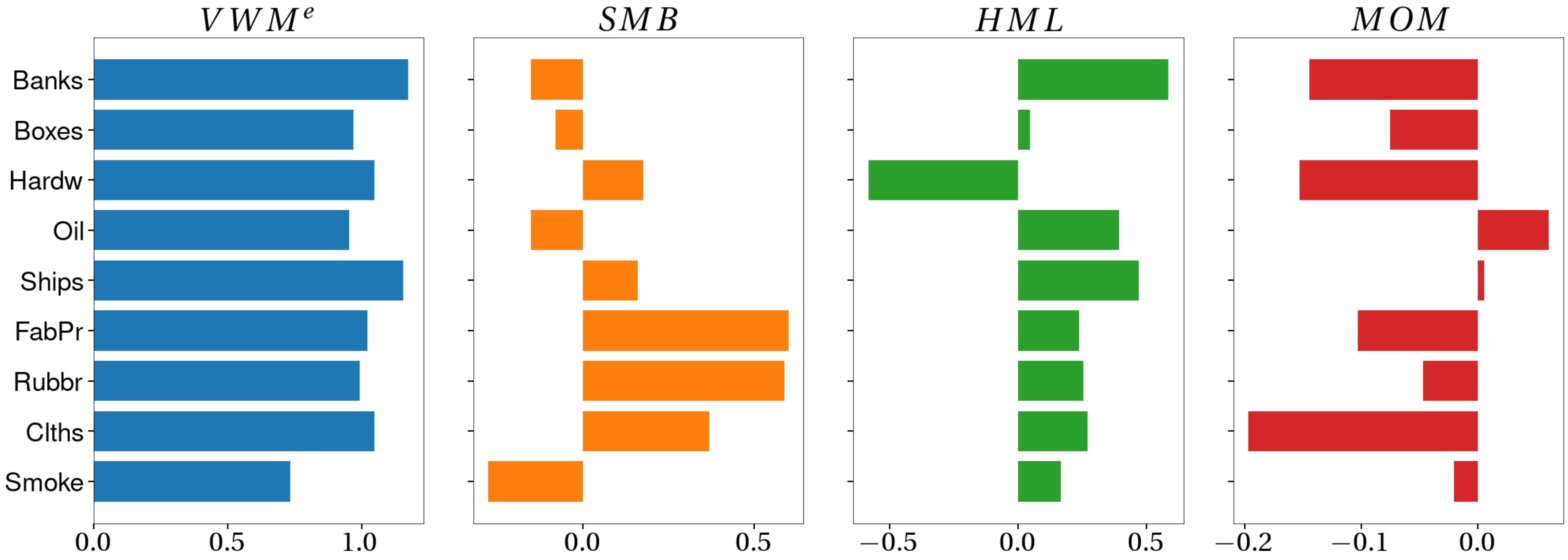


Typical Regression Coefficients

Industry Portfolios

In [22]:

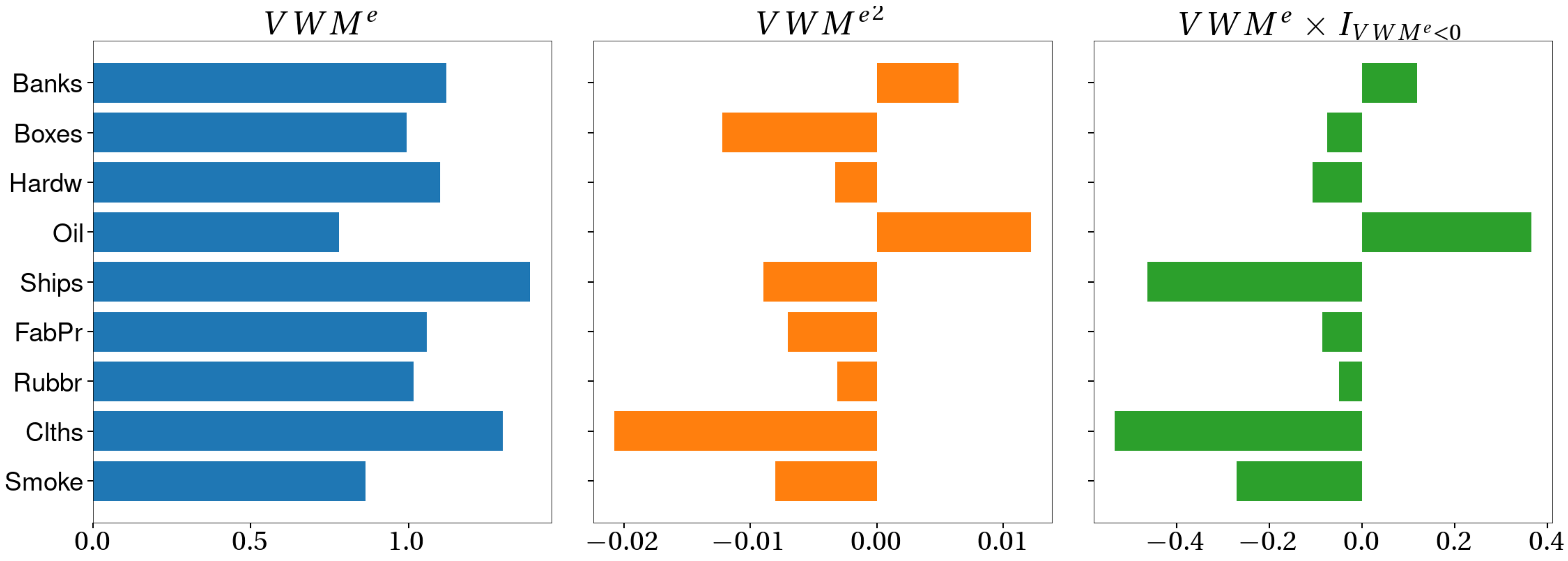
```
beta_plot(betas, titles)
```



Evidence of Non-linear returns

- Add square and asymmetry to 4-factor model

In [24]:
```
beta_plot(betas, titles)
```



Measuring fit

- Coefficient of Determination

$$R^2 = 1 - \frac{SSE}{TSS} = \frac{RSS}{TSS}$$

- Based on a complete decomposition $TSS = SSE + RSS$
- Total Sum of Squares

$$TSS = \sum_{i=1}^{n} (Y_i - \bar{Y})^2$$

- Sum of Squared Errors

$$SSE = \sum_{i=1}^{n} \hat{\epsilon}_i^2$$

- Regression Sum of Squares

$$RSS = \sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2 = \sum_{i=1}^{n} (\mathbf{x}_i \hat{\boldsymbol{\beta}} - \bar{Y})^2$$

In [25]:

```
ls = smf.ols("BHe ~ 1 + VWMe + SMB + HML + MOM", data).fit(cov_type="HC0")
summary(ls, [0])
```

OLS Regression Results

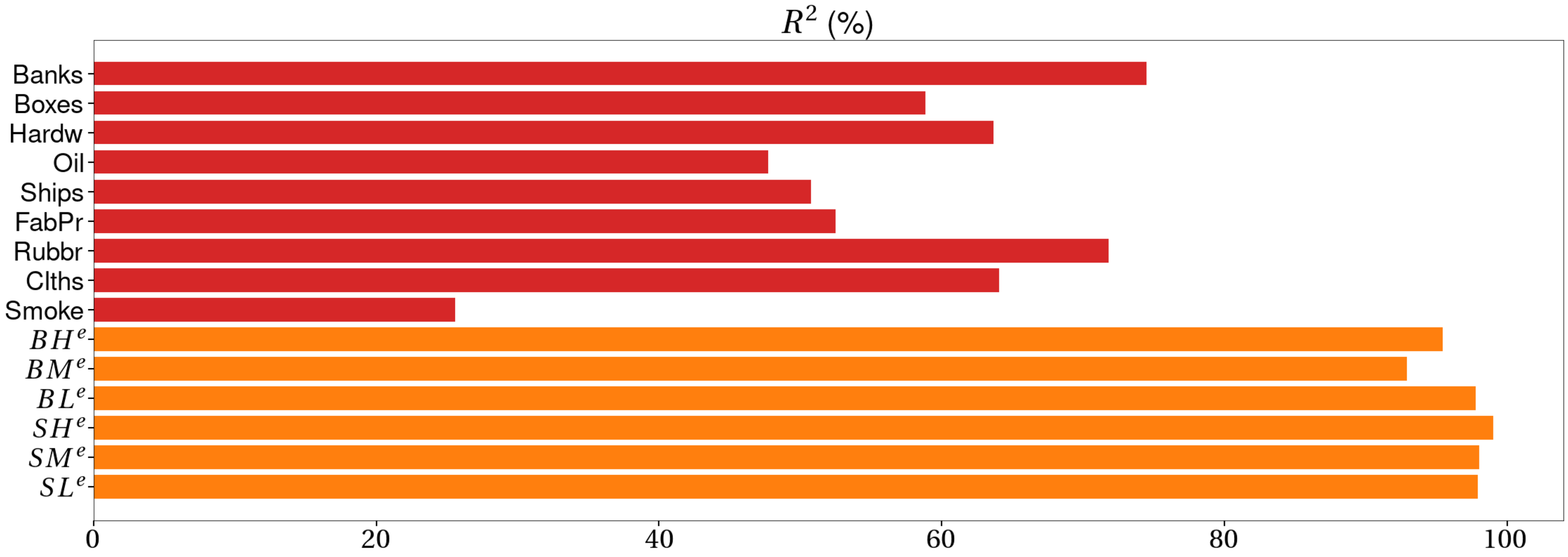
Dep. Variable:	BHe	**R-squared:**	0.954
Model:	OLS	**Adj. R-squared:**	0.954

Measuring Fit

Component and Industry Fits

In [27]:

```
r2_plot()
```



Measuring Fit

Shifting variables

$$BH_i^e + 99 = \beta_1 + \beta_2 VWM_i^e + \beta_3 SMB_i + \beta_4 HML_i + \beta_5 MOM_i + \epsilon_i$$

In [28]:

```
ls_shift = smf.ols("I(BHe + 99) ~ 1 + VWMe + SMB + HML + MOM", data).fit(cov_type="HC0")
summary(ls_shift, [0])
```

OLS Regression Results

Dep. Variable:	I(BHe + 99)	**R-squared:**	0.954
Model:	OLS	**Adj. R-squared:**	0.954

In [29]:

```
summary(ls, [0])
```

OLS Regression Results

Dep. Variable:	BHe	**R-squared:**	0.954
Model:	OLS	**Adj. R-squared:**	0.954

Measuring Fit

Rescaling variables

$$\pi BH_i^e = \beta_1 + \beta_2 VWM_i^e + \beta_3 SMB_i + \beta_4 HML_i + \beta_5 MOM_i + \epsilon_i$$

In [30]:

```
ls_scale = smf.ols("I(np.pi * BHe) ~ 1 + VWMe + SMB + HML + MOM", data).fit(
    cov_type="HC0"
)
summary(ls_scale, [0])
```

OLS Regression Results

Dep. Variable:	I(np.pi * BHe)	**R-squared:**	0.954
Model:	OLS	**Adj. R-squared:**	0.954

In [31]:

```
summary(ls, [0])
```

OLS Regression Results

Dep. Variable:	BHe	**R-squared:**	0.954
Model:	OLS	**Adj. R-squared:**	0.954

Measuring Fit

Changing the LHS Variable

$$(BH_i^e - VWM_i^e - HML_i) = \beta_1 + \beta_2 VWM_i^e + \beta_3 SMB_i + \beta_4 HML_i + \beta_5 MOM_i + \epsilon_i$$

In [32]:

```
model = "I(BHe - VWMe - HML) ~ 1 + VWMe + SMB + HML + MOM"
ls_lhs = smf.ols(model, data).fit(cov_type="HC0")
summary(ls_lhs, [0])
```

OLS Regression Results

Dep. Variable:	I(BHe - VWMe - HML)	**R-squared:**	0.382
Model:	OLS	**Adj. R-squared:**	0.378

In [33]:

```
summary(ls, [0])
```

OLS Regression Results

Dep. Variable:	BHe	**R-squared:**	0.954
Model:	OLS	**Adj. R-squared:**	0.954

Measuring fit

Caveats when model excludes the constant

$$BH_i^e + 99 = \beta_1 VWM_i^e + \beta_2 SMB_i + \beta_3 HML_i + \beta_4 MOM_i + \epsilon_i$$

In [34]:

```
ls_p99 = smf.ols("I(BHe + 99) ~ VWMe + SMB + HML + MOM - 1", data).fit(cov_type="HC0")
summary(ls_lhs, [0, 1])
```

OLS Regression Results

Dep. Variable:	I(BHe - VWMe - HML)	**R-squared:**	0.382
Model:	OLS	**Adj. R-squared:**	0.378

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.0859	0.043	-1.991	0.046	-0.170	-0.001
VWMe	0.0798	0.012	6.910	0.000	0.057	0.102
SMB	0.0019	0.017	0.110	0.912	-0.032	0.036
HML	-0.2357	0.021	-11.219	0.000	-0.277	-0.195
MOM	-0.0354	0.013	-2.631	0.009	-0.062	-0.009

Estimating the residual variance

Small-sample corrected estimator

- Variance of shock estimated using model residuals

$$s^2 = \frac{1}{n-k}\sum_{i=1}^{n}\hat{\epsilon}_i^2 = \frac{\hat{\boldsymbol{\epsilon}}'\hat{\boldsymbol{\epsilon}}}{n-k}$$

In [36]:

```
pretty(eps.T @ eps / (n - k))
```

1.1318069626411307

Large-sample estimator

- Asymptotic results usually use the large-sample version of the variance estimator

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}\hat{\epsilon}_i^2 = \frac{\hat{\boldsymbol{\epsilon}}'\hat{\boldsymbol{\epsilon}}}{n}$$

In [37]:
```
pretty(eps.T @ eps / n)
```

1.123557640755991

Scores and the first-order condition of OLS

- The FOC of a regression is

$$\mathbf{X}'\hat{\boldsymbol{\epsilon}} = \sum_{i=1}^{n} \mathbf{x}_i'\hat{\epsilon}_i = \mathbf{0}$$

- Estimated residuals are *always* orthogonal with included regressors
- Later we will see these can be used to test models if $\approx \mathbf{0}$

In [39]:
```
scores
```

Out[39]:

	Scores
Intercept	7.056578e-13
VWMe	1.274314e-11
SMB	5.496090e-13
HML	3.228973e-12
MOM	-2.858463e-12

Analysis of Cross-Sectional Data

Properties of OLS Estimators

- Invariance to Affine Transformations
- Asymptotic Distribution
- Feasible Central Limit Theorems
- Bootstrap Estimation of the Covariance

Variable Transformations

Rescaling by a constant

$$\frac{Y_i}{100} = \beta_1 + \beta_2 \frac{X_{i,2}}{100} + \ldots + \beta_k \frac{X_{i,k}}{100} + \epsilon_i$$

In [40]:

```
model = "BHe ~ 1 + VWMe + SMB + HML + MOM"
rescaled_ls = smf.ols(model, data / 100.0).fit(cov_type="HC0")
show_params(rescaled_ls, ls, columns=["Rescaled", "Orig"])
```

Out[40]:

	Rescaled	Orig
Intercept	-0.000859	-0.085899
VWMe	1.079785	1.079785
SMB	0.001894	0.001894
HML	0.764300	0.764300
MOM	-0.035397	-0.035397

Variable Transformations

Rescaling single variables

$$Y_i = \beta_1 + \beta_2 \left(2VWM_i^e\right) + \beta_3 SMB_i + \beta_4 \frac{HML_i}{2} + \beta_4 MOM_i + \epsilon_i$$

In [41]:

```
model = "BHe ~ 1 + I(2 * VWMe) + SMB + I(1/2 * HML) + MOM"
ls_p10 = smf.ols(model, data).fit(cov_type="HC0")
show_params(ls_p10, columns=["Plus 10"])
```

Out[41]:

	Plus 10
Intercept	-0.085899
I(2 * VWMe)	0.539893
SMB	0.001894
I(1 / 2 * HML)	1.528600
MOM	-0.035397

Variable Transformations

Affine Transformations

$$(3BH^e + 7) = \beta_1 + \beta_2 \left(2VWM_i^e - 9\right) + \beta_3 \frac{SMB_i}{2} + \beta_4 HML_i + \beta_4 MOM_i + \epsilon_i$$

In [42]:

```
model = "I(3 * BHe + 7)  ~ 1 + I(2 * VWMe - 9) + I(1/2 *SMB) + HML + MOM"
ls_affine = smf.ols(model, data).fit(cov_type="HC0")
show_params(ls_affine, columns=["Affine"])
```

Out[42]:

	Affine
Intercept	21.319408
I(2 * VWMe - 9)	1.619678
I(1 / 2 * SMB)	0.011361
HML	2.292901
MOM	-0.106192

In [43]:

```
pretty(f"The ratio is {ls_affine.params['I(2 * VWMe - 9)'] / ls.params['VWMe']:0.3f}")
```

The ratio is 1.500

Characterizing Parameter Estimation Error

- Central Limit Theorem

$$\sqrt{n}\left(\hat{\boldsymbol{\beta}}_n - \boldsymbol{\beta}\right) \xrightarrow{d} N\left(\mathbf{0}, \boldsymbol{\Sigma}_{XX}^{-1}\mathbf{S}\boldsymbol{\Sigma}_{XX}^{-1}\right)$$

- Covariance components $\boldsymbol{\Sigma}_{XX} = E\left[\mathbf{x}_i'\mathbf{x}_i\right]$ and $\mathbf{S} = \mathrm{p} -$ $\lim_{n\to\infty} \mathrm{Var}\left[\sqrt{n}\frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i'\epsilon_i\right]$.

- In practice

$$\hat{\boldsymbol{\beta}}_n \overset{\approx}{\sim} N\left(\boldsymbol{\beta}, \frac{\hat{\boldsymbol{\Sigma}}_{XX}^{-1}\hat{\mathbf{S}}\hat{\boldsymbol{\Sigma}}_{XX}^{-1}}{n}\right)$$

In [44]:

```
ls.cov_params()
```

Out[44]:

	Intercept	VWMe	SMB	HML	MOM
Intercept	0.001860	-0.000171	0.000079	-0.000157	-0.000154
VWMe	-0.000171	0.000133	-0.000060	0.000039	0.000019
SMB	0.000079	-0.000060	0.000297	0.000042	0.000019
HML	-0.000157	0.000039	0.000042	0.000441	0.000122
MOM	-0.000154	0.000019	0.000019	0.000122	0.000181

Characterizing Parameter Estimation Error

Estimating the Covariance

$$\hat{\mathbf{\Sigma}}_{XX} = \frac{1}{n}\mathbf{X}'\mathbf{X} \text{ and } \hat{\mathbf{S}} = \sum_{i=1}^{n} \hat{\epsilon}_i{}^2 \mathbf{x}_i'\mathbf{x}_i$$

In [45]:

```
xe = x * eps
S = xe.T @ xe / n
Sigma = x.T @ x / n
Sigma_inv = np.linalg.inv(Sigma)
cov = 1 / n * (Sigma_inv @ S @ Sigma_inv)
cov.columns = x.columns
cov.index = x.columns
cov
```

Out[45]:

	Intercept	VWMe	SMB	HML	MOM
Intercept	0.001860	-0.000171	0.000079	-0.000157	-0.000154
VWMe	-0.000171	0.000133	-0.000060	0.000039	0.000019
SMB	0.000079	-0.000060	0.000297	0.000042	0.000019
HML	-0.000157	0.000039	0.000042	0.000441	0.000122
MOM	-0.000154	0.000019	0.000019	0.000122	0.000181

Characterizing Parameter Estimation Error

Standard Errors

- Root of diagonal elements of VCV

In [46]:

```
pretty(ls.bse)
```

Out[46]:

Intercept	0.043134
VWMe	0.011547
SMB	0.017224
HML	0.021009
MOM	0.013455

Bootstrapping the Covariance

- Simulate from data to estimate covariance
- Randomly sample n observation with replacement $(y_i, \mathbf{x}_i)$
- Estimate $\hat{\beta}_b$ from random sample
- Repeat B times
- Compute covariance from bootstrapped $\hat{\beta}_b$

In [47]:

```
betas = []
g = np.random.default_rng(2020)
for i in range(100):
    idx = g.integers(n, size=n)
    xb = x.iloc[idx]
    y = xb @ ls.params + eps[idx, 0]
    beta = sm.OLS(y, xb).fit().params
    betas.append(beta)
betas = np.array(betas)

betas = pd.DataFrame(betas, columns=x.columns)
```

In [48]:

```
betas.cov()
```

Out[48]:

	Intercept	VWMe	SMB	HML	MOM
Intercept	0.001791	-0.000111	0.000246	-0.000082	-0.000127
VWMe	-0.000111	0.000099	-0.000066	-0.000005	0.000011
SMB	0.000246	-0.000066	0.000355	0.000079	0.000029
HML	-0.000082	-0.000005	0.000079	0.000462	0.000121
MOM	-0.000127	0.000011	0.000029	0.000121	0.000208

In [49]:

```
ls.cov_params()
```

Out[49]:

	Intercept	VWMe	SMB	HML	MOM
Intercept	0.001860	-0.000171	0.000079	-0.000157	-0.000154
VWMe	-0.000171	0.000133	-0.000060	0.000039	0.000019
SMB	0.000079	-0.000060	0.000297	0.000042	0.000019
HML	-0.000157	0.000039	0.000042	0.000441	0.000122
MOM	-0.000154	0.000019	0.000019	0.000122	0.000181

Analysis of Cross-Sectional Data

Wald and t-tests

- Linear Equality Hypotheses
- Testing a Single Restriction with a t-tests
- The t-statistic
- Multiple Restrictions and the Wald tests
- The F-stats

Hypothesis Testing

- Null in a Linear Equality Test

$$H_0 : \mathbf{R}\beta = r$$

- Three classes of tests
    - Wald and t-test
    - Lagrange Multiplier
    - Likelihood Ratio

Hypothesis Testing

t-tests

- Asymptotically normally distributed
- Test a single restriction
- Values outside of $\pm 1.96 \approx \pm 2$ lead to rejection using a 5% size
- Can be used to test 1-sided hypotheses

Hypothesis Testing

t-test Example

Testing the additional total effect is 0

$$H_0 : SMB + HML + MOM = 0$$

$$R = [0, 0, 1, 1, 1], r = 0$$

In [50]:

```
summary(ls)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.0859	0.043	-1.991	0.046	-0.170	-0.001
VWMe	1.0798	0.012	93.514	0.000	1.057	1.102
SMB	0.0019	0.017	0.110	0.912	-0.032	0.036
HML	0.7643	0.021	36.380	0.000	0.723	0.805
MOM	-0.0354	0.013	-2.631	0.009	-0.062	-0.009

Hypothesis Testing

t-test Example

In [51]:

```
R = np.array([[0, 0, 1, 1, 1]])
c = ls.cov_params()
h0_vcv = np.squeeze(R @ c @ R.T)
t = (R @ ls.params) / np.sqrt(h0_vcv)
pretty(t[0])
```

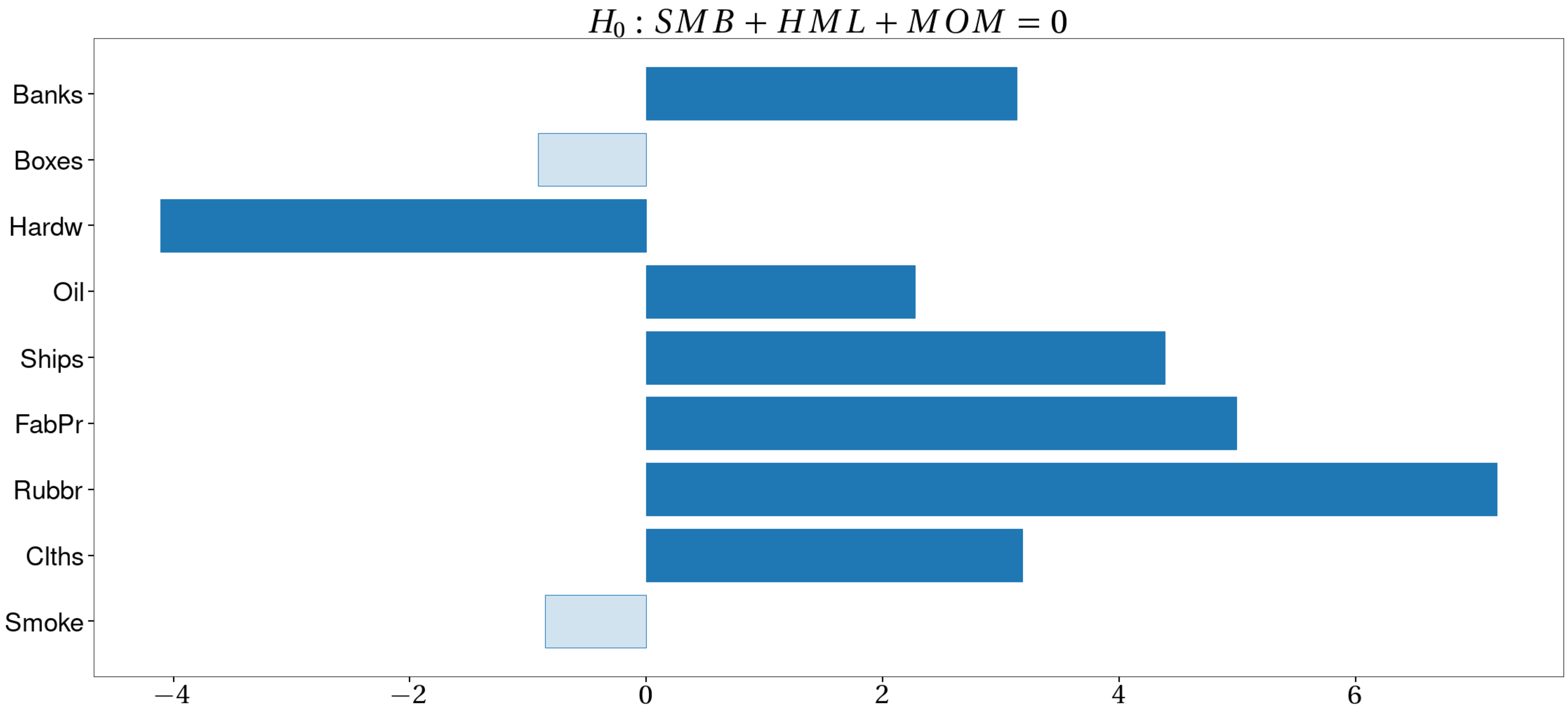
20.385884770084246

Hypothesis Testing

t-test Example on Industry Portfolios

In [53]:

```
test_plot(t_tests, title="$H_0: SMB + HML + MOM = 0$")
```



Hypothesis Testing

t-stats

- t-stat is special case for $H_0 : \beta_j = 0$
- Most commonly reported test statistic
- Asymptotic normal
- 5% critical values $\pm 1.96 \approx \pm 2$

In [54]:

```
pretty(ls.tvalues)
```

Out[54]:

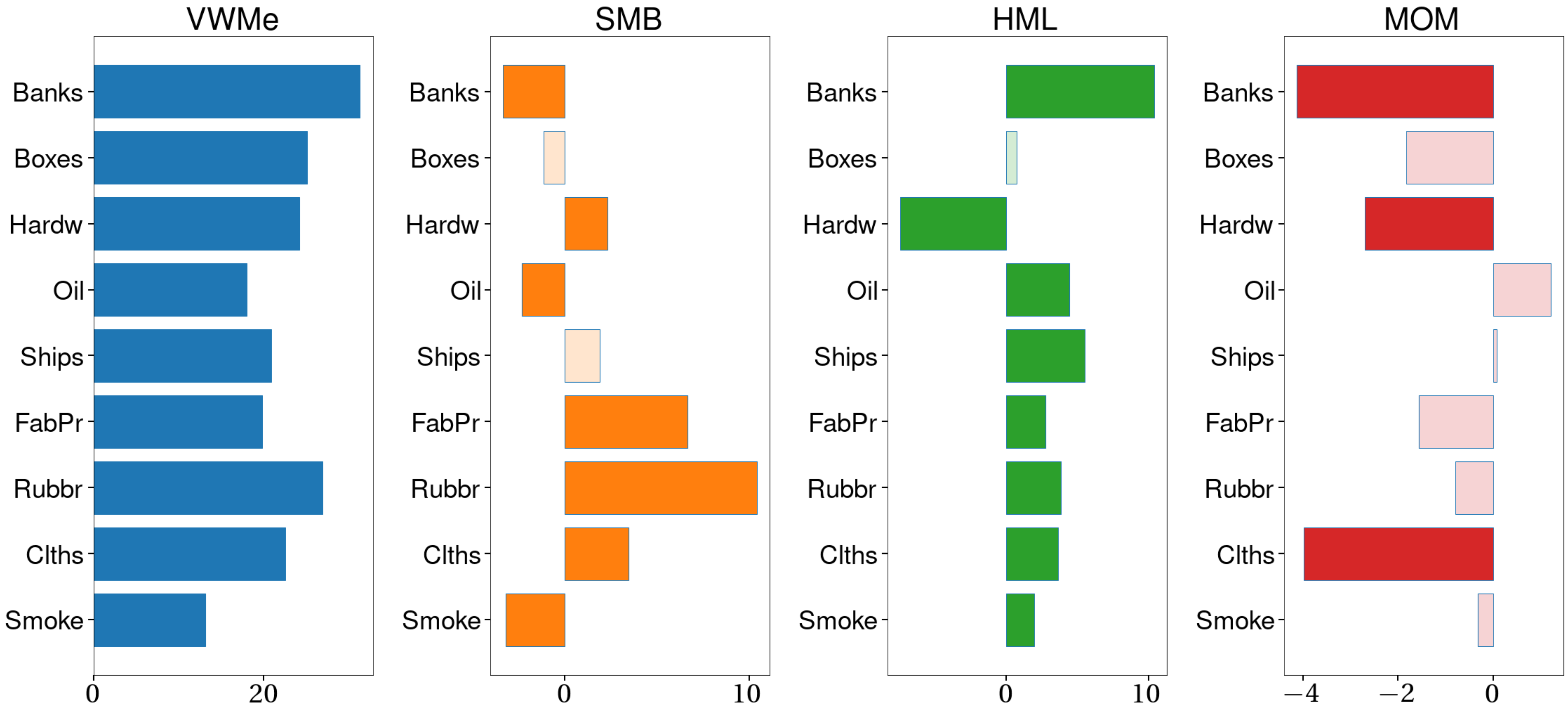
Intercept	-1.991463
VWMe	93.513503
SMB	0.109934
HML	36.380381
MOM	-2.630803

Hypothesis Testing

Significance in Industry Portfolios

In [56]:

```
multi_test_plot(t_stats)
```


VWMe
SMB
HML
MOM
Banks
Boxes
Hardw
Oil
Ships
FabPr
Rubbr
Clths
Smoke
0
20
10
−4
−2

Hypothesis Testing

Wald Tests

- Test multiple hypothesis
- Exploit properties of multivariate normals
- χ^2_m distributed in large samples
- Test statistic is

$$W = n\left(\mathbf{R}\hat{\boldsymbol{\beta}} - \mathbf{r}\right)'\left[\mathbf{R}\hat{\boldsymbol{\Sigma}}_{XX}^{-1}\hat{\mathbf{S}}\hat{\boldsymbol{\Sigma}}_{XX}^{-1}\mathbf{R}'\right]^{-1}\left(\mathbf{R}\hat{\boldsymbol{\beta}} - \mathbf{r}\right)$$

Hypothesis Testing

Wald Tests

Example

- Multiple β all zero: $H_0 : SMB = HML = MOM = 0$

$$R = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, r = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

In [57]:

```
R = np.zeros((3, 5))
R[0, 2] = R[1, 3] = R[2, 4] = 1
r = np.zeros(3)
h0_vcv = R @ c @ R.T
h0_vcv.columns = h0_vcv.index = [f"Restr {i}" for i in range(1, 4)]
h0_vcv
```

Out[57]:

	Restr 1	Restr 2	Restr 3
Restr 1	0.000297	0.000042	0.000019
Restr 2	0.000042	0.000441	0.000122
Restr 3	0.000019	0.000122	0.000181

In [58]:

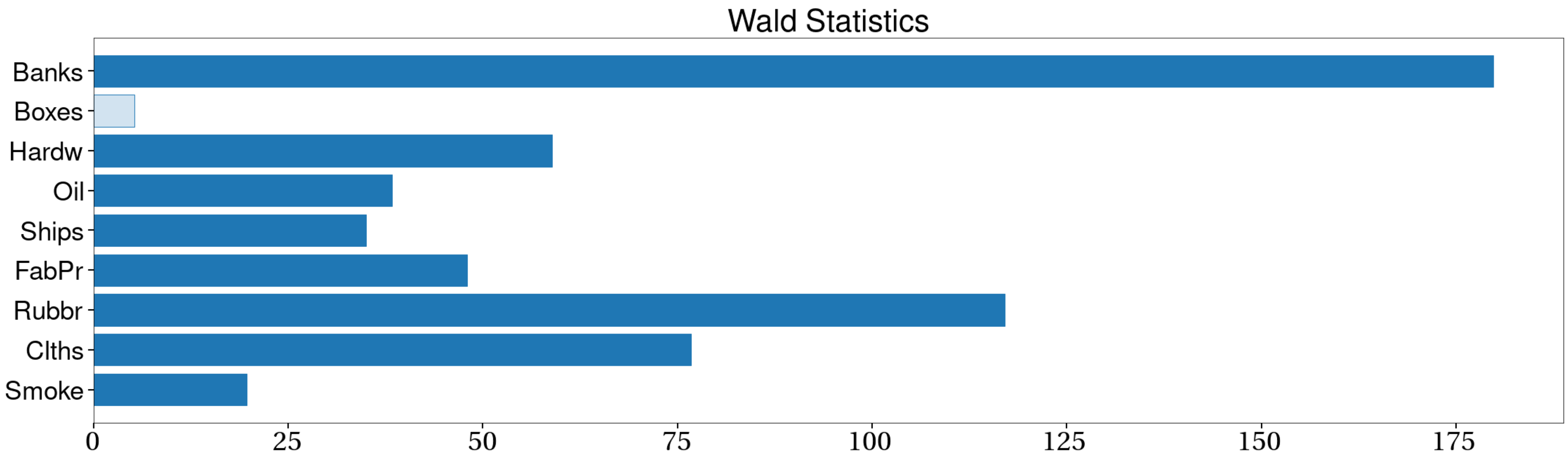
```
numerator = R @ ls.params - r
wald = numerator @ np.linalg.inv(h0_vcv) @ numerator.T
pretty(f"W={wald:0.1f}")
```

W=1749.6

In [60]:

```
dof = 3
pretty(f"The critical value is {stats.chi2(dof).ppf(0.95):0.2f} from a $\chi^2_{dof}$")
test_plot(walds, cv=stats.chi2(dof).ppf(0.95), title="Wald Statistics")
```

The critical value is 7.81 from a χ^2_3



Hypothesis Testing

The F-stat

- Special case of Wald for

$$H_0 : \beta_2 = \beta_3 = \ldots = \beta_k = 0$$

- Never test constant
    - **Note**: If no constant

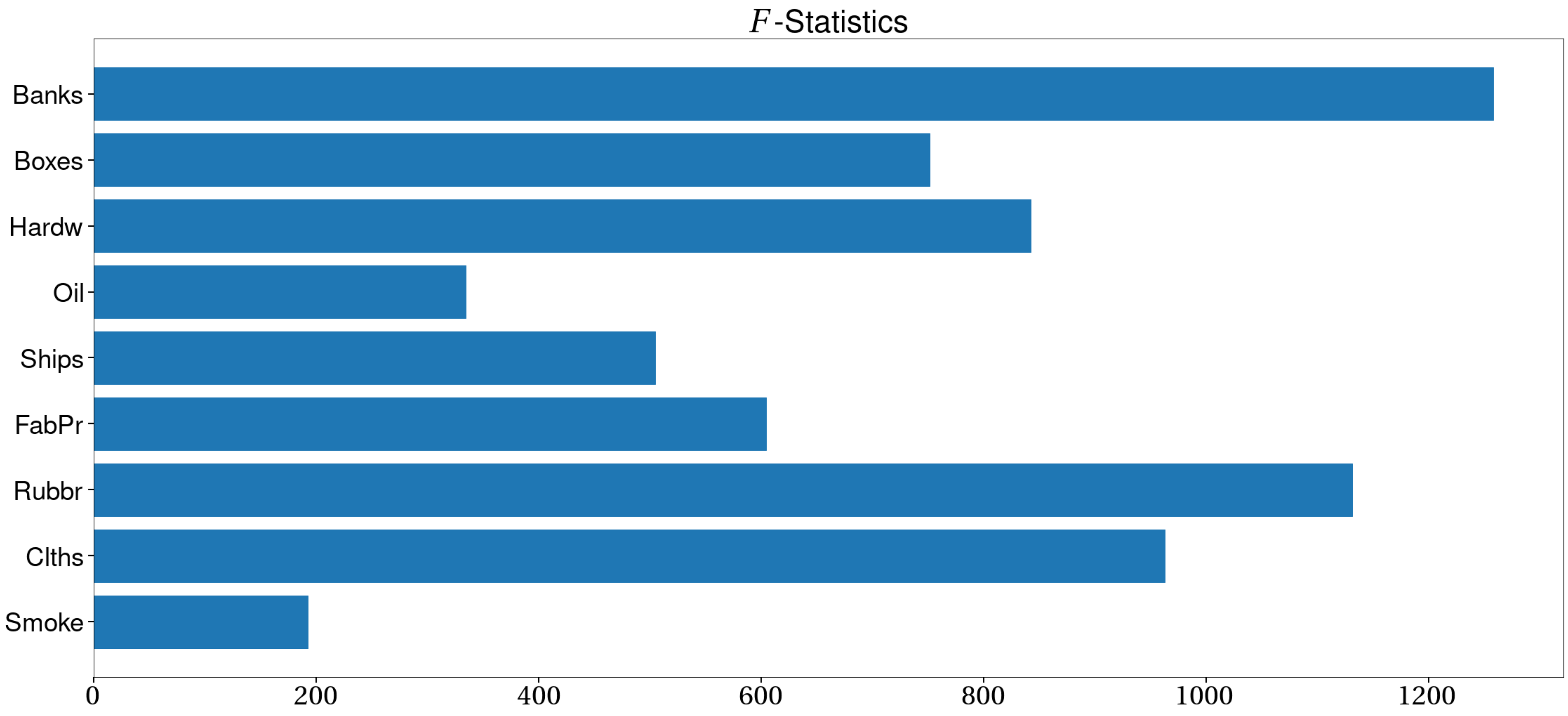
$$H_0 : \beta_1 = \beta_2 = \ldots = \beta_k = 0$$

- Example restrictions

$$R = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, r = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

In [62]:
```
dof = 4
cv = stats.chi2(dof).ppf(0.95)
pretty(f"The critical value is {cv:0.2f} from a $\chi^2_{dof}$")
test_plot(f_stats, cv=cv, title="$F$-Statistics")
```

The critical value is 9.49 from a χ^2_4



Analysis of Cross-Sectional Data

LM and LR Tests

- Imposing a LER on a Linear Regression
- LM Tests
- LR Tests
- Comparing Wald, LM and LR tests

Hypothesis Testing

Imposing the null on the model

$$H_0 : \beta_{SMB} + \beta_{HML} + \beta_{MOM} = 0 \Rightarrow \beta_{SMB} = -\beta_{HML} - \beta_{MOM}$$

Initial model

$$Ships = \beta_1 + \beta_2 VWM^e + \beta_3 SMB + \beta_4 HML + \beta_5 MOM + \epsilon_i$$

becomes

$$Ships = \beta_1 + \beta_2 VWM^e + (-\beta_4 - \beta_5) SMB + \beta_4 HML + \beta_5 MOM + \epsilon_i$$

and then finally

$$Ships = \beta_1 + \beta_2 VWM^e + \beta_4 (HML - SMB) + \beta_5 (MOM - SMB) + \epsilon_i$$

In [63]:

```
model = "Ships ~ 1 + VWMe + I(HML-SMB) + I(MOM-SMB)"
imposed = smf.ols(model, data).fit(cov_type="HC0")
summary(imposed)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	0.0676	0.202	0.335	0.738	-0.328	0.463
VWMe	1.1401	0.058	19.805	0.000	1.027	1.253
I(HML - SMB)	0.1897	0.061	3.133	0.002	0.071	0.308
I(MOM - SMB)	-0.1304	0.061	-2.148	0.032	-0.249	-0.011

Hypothesis Testing

Lagrange Multiplier (LM) tests

- Uses property that scores should be 0 when model is correct
- Define scores

$$s_i = \mathbf{x}_i \tilde{\epsilon}_i$$

- Estimate score covariance using

$$\hat{\tilde{S}} = \frac{1}{n} \sum_{i=1}^{n} s_i' s_i$$

- LM test statistic is defined

$$LM = n \bar{s} \hat{\tilde{S}} \bar{s}' \xrightarrow{d} \chi^2_m$$

In [64]:

```
imposed_eps = imposed.resid.to_numpy()
scores = x * imposed_eps[:, None]
mean_scores = scores.mean()
pretty(mean_scores)
```

Out[64]:

Intercept	-1.437140e-16
VWMe	1.680030e-14
SMB	1.614948e+00
HML	1.614948e+00
MOM	1.614948e+00

In [65]:

```
S = scores.T @ scores / n
LM = n * mean_scores @ np.linalg.inv(S) @ mean_scores
LM
```

Out[65]:

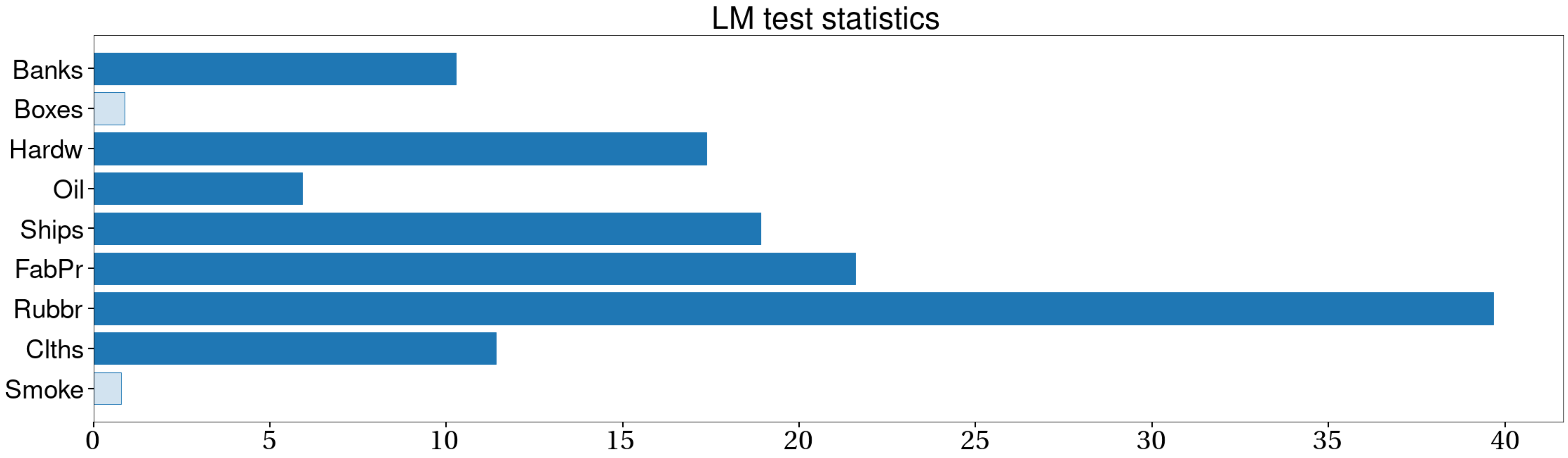
```
18.912996411664253
```

Hypothesis Testing

LM Tests on Industry Portfolios

In [67]:
```
dof = 1
cv = stats.chi2(dof).ppf(0.95)
pretty(f"The critical value is {cv:0.2f} from a $\chi^2_{dof}$")
test_plot(lms, cv=cv, title="LM test statistics")
```

The critical value is 3.84 from a χ^2_1



Hypothesis Testing

Likelihood Ratio (LR) tests

- Nearly identical to LM, only using unrestricted model to estimate score covariance

$$\hat{s}_i = \mathbf{x}_i \hat{\epsilon}_i$$

- Covariance uses $\hat{\epsilon}_i$ instead of $\tilde{\epsilon}_i$

$$\hat{S} = \frac{1}{n} \sum_{i=1}^{n} s_i' s_i$$

- LR test statistic is defined

$$LR = n \bar{s} \hat{S} \bar{s}' \xrightarrow{d} \chi^2_m$$

In [68]:

```python
unres = smf.ols("Ships ~ 1 + VWMe + SMB + HML + MOM", data).fit()
eps = unres.resid.to_numpy()
s_hat = x * eps[:, None]
S = s_hat.T @ s_hat / n
LR = n * mean_scores @ np.linalg.inv(S) @ mean_scores
LR
```

Out[68]:

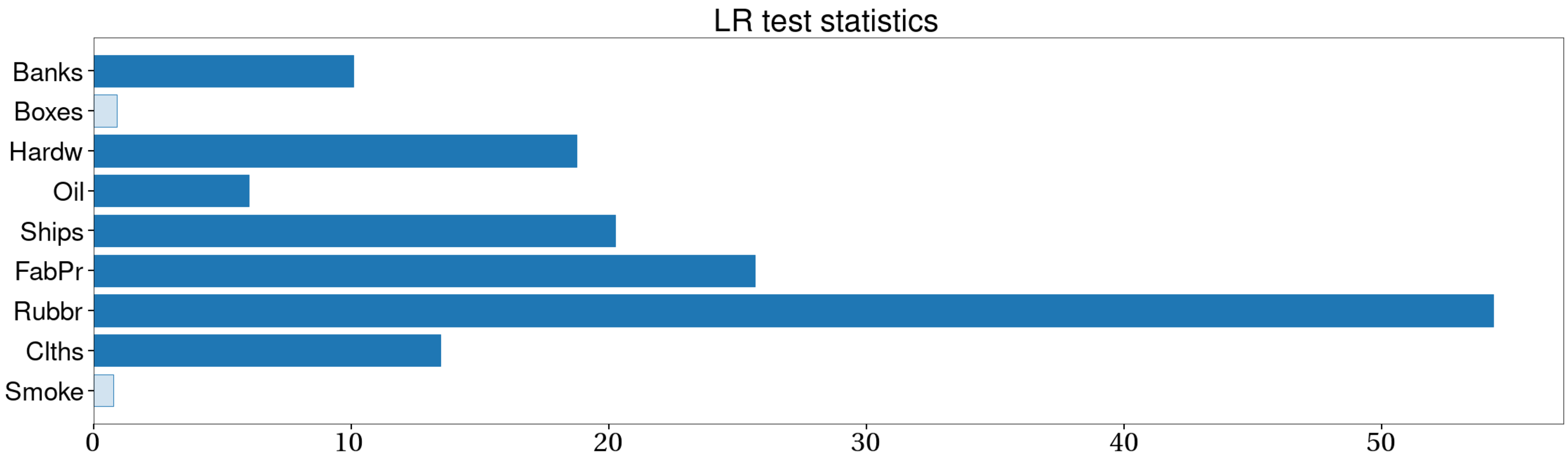
```
4.189010948813025
```

Hypothesis Testing

LM Tests on Industry Portfolios

In [70]:

```
test_plot(lrs, cv=cv, title="LR test statistics")
```

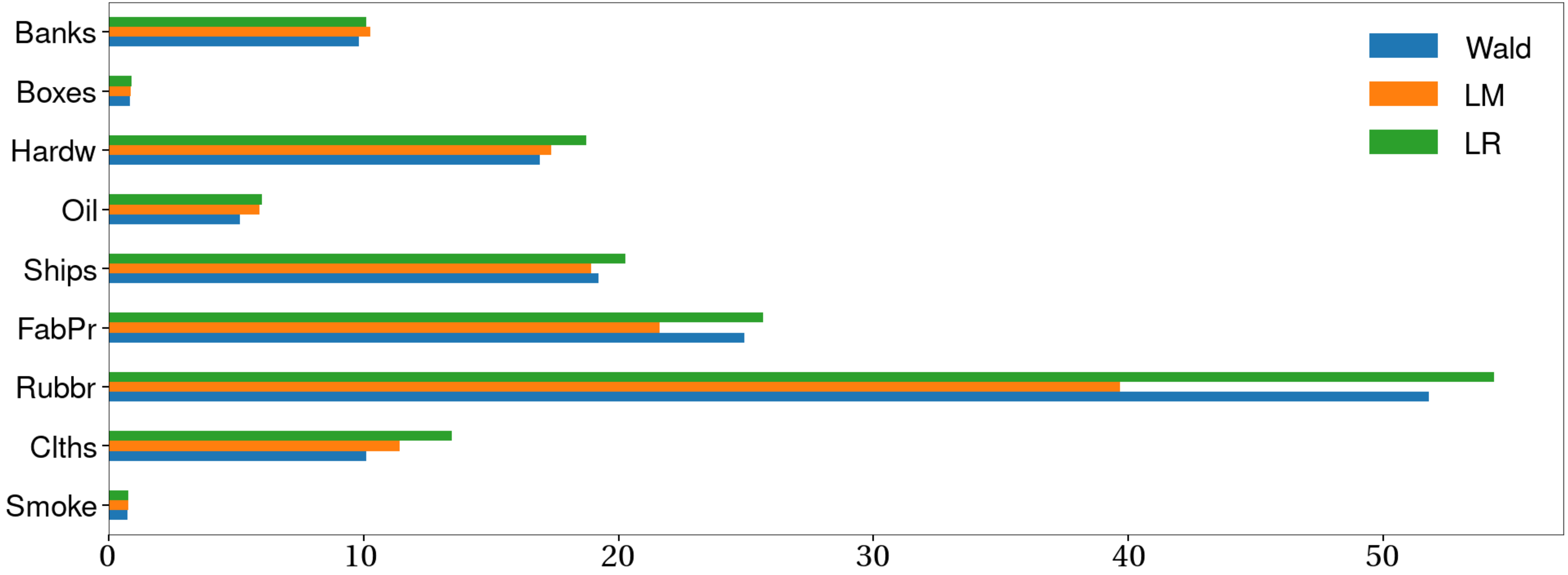


Hypothesis Testing

Comparing the Three Classes of Test

In [72]:

```
plot_three_stats()
```



Analysis of Cross-Sectional Data

Heteroskedasticity

- Testing for Heteroskedasticity
- Covariance Estimation for Homoskedastic Data
- Boostrap Covariance Estimation for Homoskedastic Data

Testing for Heteroskedasticity

White's test

- Key insight of White: Heteroskedasticity robust estimator only needed when

$$E[\epsilon_i^2 X_{i,o} X_{i,p}] \neq \sigma^2 E[X_{i,o} X_{i,p}]$$

- Use a regression to test if covariances are all 0

$$\hat{\epsilon}_i^2 = \mathbf{z}_i \boldsymbol{\gamma} + \eta_i$$

- $\mathbf{z}$ contains all distinct cross-products of X
- Test statistic is nR^2 from auxiliary model
- $\chi^2_{k(k+1)/2-1}$ distribution when initial model includes a constant

In [73]:

```python
data["eps2"] = ls.resid ** 2

crosses = ""
for x1 in ("VWMe", "SMB", "HML", "MOM"):
    for x2 in ("VWMe", "SMB", "HML", "MOM"):
        crosses += f"+ I({x1} * {x2})"
formula = "eps2 ~ 1 + VWMe + SMB + HML + MOM " + crosses
pretty(formula)
```

eps2 ~ 1 + VWMe + SMB + HML + MOM + I(VWMe * VWMe)+ I(VWMe * SMB)+ I(VWMe * HML)+ I(VWMe * MOM)+ I(SMB * VWMe)+ I(SMB * SMB)+ I(SMB * HML)+ I(SMB * MOM)+ I(HML * VWMe)+ I(HML * SMB)+ I(HML * HML)+ I(HML * MOM)+ I(MOM * VWMe)+ I(MOM * SMB)+ I(MOM * HML)+ I(MOM * MOM)

In [74]:

```
white = smf.ols(formula, data).fit()
summary(white, [0])
```

OLS Regression Results

Dep. Variable:	eps2	**R-squared:**	0.109
Model:	OLS	**Adj. R-squared:**	0.090

In [76]:

```
summary_white()
```

	Intercept	VWMe	SMB	HML	MOM	VWMe * VWMe	VWMe * SMB	VWMe * HML	VWMe * MOM	SMB * VWMe	SMB * SMB
Parameter	8.134412e-01	-0.044663	0.063097	0.015792	0.009063	0.003135	-0.003666	-0.002738	-0.001518	-0.003666	0.006417
t-test Stat.	8.358941e+00	-2.275155	2.089132	0.535791	0.431262	1.105676	-1.284523	-1.076393	-0.817672	-1.284523	1.354750
p-value	3.645328e-16	0.023211	0.037072	0.592281	0.666416	0.269263	0.199402	0.282138	0.413835	0.199402	0.175953

	SMB * HML	SMB * MOM	HML * VWMe	HML * SMB	HML * HML	HML * MOM	MOM * VWMe	MOM * SMB	MOM * HML	MOM * MOM
Parameter	0.005350	0.003334	-0.002738	0.005350	0.023196	0.010208	-0.001518	0.003334	0.010208	0.003795
t-test Stat.	1.210476	1.313499	-1.076393	1.210476	4.640325	4.133473	-0.817672	1.313499	4.133473	2.079526
p-value	0.226522	0.189464	0.282138	0.226522	0.000004	0.000040	0.413835	0.189464	0.000040	0.037948

In [77]:

```
white_stat = n * white.rsquared
pretty(f"White's stat: {white_stat:0.2f}")
```

White's stat: 74.76

In [78]:

```
pretty(f"Number of restrictions: {5 * (5 + 1) // 2 + 1}")
```

Number of restrictions: 16

In [79]:

```
pvalue = 1.0 - stats.chi2(5 * (5 + 1) / 2 + 1).cdf(white_stat)
pretty(f"The p-value is {pvalue:0.3f}")
```

The p-value is 0.000

Characterizing Parameter Estimation Error

Homoskedastic Data

- Central Limit Theorem when residuals are homoskedastic

$$\sqrt{n}\left(\hat{\boldsymbol{\beta}}_n - \boldsymbol{\beta}\right) \xrightarrow{d} N\left(\mathbf{0}, \sigma^2 \mathbf{\Sigma}_{XX}^{-1}\right)$$

- Covariance components $\mathbf{\Sigma}_{XX} = E\left[\mathbf{x}_i'\mathbf{x}_i\right]$ and $\sigma^2 = E[\epsilon_i^2]$.

- In practice

$$\hat{\boldsymbol{\beta}}_n \stackrel{\approx}{\sim} N\left(\boldsymbol{\beta}, \frac{\hat{\sigma}^2 \hat{\mathbf{\Sigma}}_{XX}^{-1}}{n}\right)$$

In [80]:
```python
ls_homo = smf.ols("BHe ~ 1 + VWMe + SMB + HML + MOM", data).fit()
summary(ls_homo)
```

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.0859	0.042	-2.038	0.042	-0.169	-0.003
VWMe	1.0798	0.010	108.710	0.000	1.060	1.099
SMB	0.0019	0.014	0.134	0.893	-0.026	0.030
HML	0.7643	0.015	50.772	0.000	0.735	0.794
MOM	-0.0354	0.010	-3.504	0.000	-0.055	-0.016

In [81]:
```python
summary(ls)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.0859	0.043	-1.991	0.046	-0.170	-0.001
VWMe	1.0798	0.012	93.514	0.000	1.057	1.102
SMB	0.0019	0.017	0.110	0.912	-0.032	0.036
HML	0.7643	0.021	36.380	0.000	0.723	0.805
MOM	-0.0354	0.013	-2.631	0.009	-0.062	-0.009

In [82]:

```
ls_homo.cov_params()
```

Out[82]:

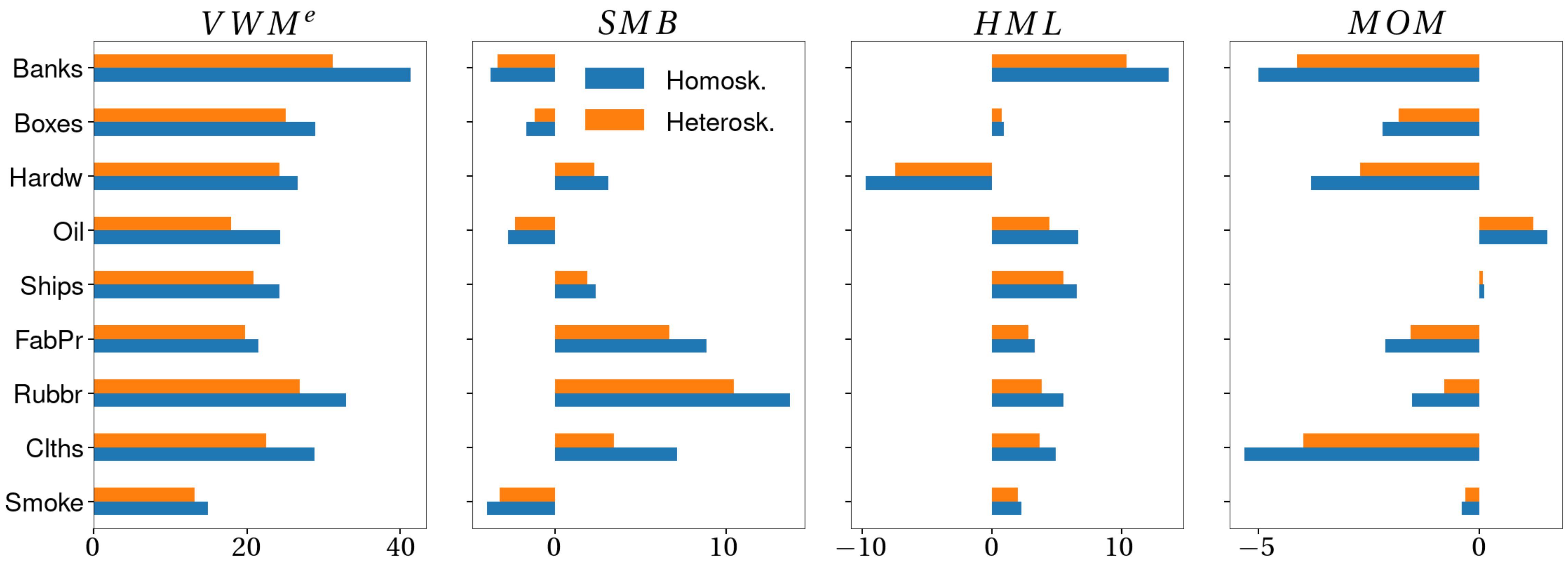
	Intercept	VWMe	SMB	HML	MOM
Intercept	0.001777	-0.000069	-2.196720e-05	-0.000104	-8.770776e-05
VWMe	-0.000069	0.000099	-3.737690e-05	0.000033	1.946526e-05
SMB	-0.000022	-0.000037	1.996325e-04	0.000024	7.559086e-07
HML	-0.000104	0.000033	2.404199e-05	0.000227	3.602389e-05
MOM	-0.000088	0.000019	7.559086e-07	0.000036	1.020344e-04

Heteroskedasticity vs Homoskedasticity

Industry Portfolios

In [84]:

```
plot_tvalues()
```



Bootstrap for Homoskedastic Data

- If data are homoskedastic can use improved bootstrap
- Independently sample $\hat{\epsilon}_i$ and $\mathbf{x}_j$ and then build simulated $\tilde{Y}_m = \mathbf{x}_j\hat{\boldsymbol{\beta}} + \hat{\epsilon}_i$
- Estimate model on bootstrapped data
- Repeat $b = 1, 2, \ldots, B$ times and compute covariance of estimated $\hat{\boldsymbol{\beta}}_b$

In [85]:

```
eps = ls.resid.to_numpy()
betas = []
g = np.random.default_rng(2020)
x = ls.model.data.orig_exog
for i in range(1000):
    x_idx = g.integers(n, size=n)
    xb = x.iloc[x_idx]
    eps_idx = g.integers(n, size=n)
    y = xb @ ls.params + eps[eps_idx]
    beta = sm.OLS(y, xb).fit().params
    betas.append(beta)
betas = np.array(betas)

betas = pd.DataFrame(betas, columns=x.columns)

betas.cov()
```

Out[85]:

	Intercept	VWMe	SMB	HML	MOM
Intercept	0.001796	-0.000077	0.000007	-0.000104	-0.000095
VWMe	-0.000077	0.000098	-0.000030	0.000032	0.000013
SMB	0.000007	-0.000030	0.000198	0.000026	0.000011
HML	-0.000104	0.000032	0.000026	0.000229	0.000040
MOM	-0.000095	0.000013	0.000011	0.000040	0.000103

Analysis of Cross-Sectional Data

Model Selection

- General-to-Specific
- Specific-to-General
- Information Criteria
- Cross-Validation

Model Selection

General-to-Specific

- Start with full model
- Drop variables one-at-a-time when P-value > α
    - Typical sizes 1% or .1%

In [86]:

```
res = smf.ols("Ships ~ 1 + VWMe + SMB + HML + MOM", data).fit(cov_type="HC0")
summary(res)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.1420	0.204	-0.697	0.486	-0.541	0.257
VWMe	1.1551	0.055	20.872	0.000	1.047	1.264
SMB	0.1604	0.084	1.898	0.058	-0.005	0.326
HML	0.4703	0.085	5.513	0.000	0.303	0.637
MOM	0.0055	0.074	0.074	0.941	-0.140	0.151

In [87]:

```
res = smf.ols("Ships ~ 1 + VWMe + SMB + HML", data).fit(cov_type="HC0")
summary(res)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.1372	0.197	-0.697	0.486	-0.523	0.248
VWMe	1.1541	0.056	20.716	0.000	1.045	1.263
SMB	0.1603	0.084	1.900	0.057	-0.005	0.326
HML	0.4683	0.086	5.474	0.000	0.301	0.636

In [88]:

```
res = smf.ols("Ships ~ 1 + VWMe + HML", data).fit(cov_type="HC0")
summary(res)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.1201	0.197	-0.610	0.542	-0.506	0.266
VWMe	1.1842	0.054	21.870	0.000	1.078	1.290
HML	0.4492	0.084	5.371	0.000	0.285	0.613

In [89]:

```
res = smf.ols("Ships ~ VWMe + HML - 1", data).fit(cov_type="HC0")
summary(res)
```

	coef	std err	z	P>|z|	[0.025	0.975]
VWMe	1.1802	0.054	22.045	0.000	1.075	1.285
HML	0.4442	0.083	5.332	0.000	0.281	0.608

Model Selection

Specific-to-General

- Start with only a constant
- Add variables one-at-a-time and keep smallest P-value if > α

In [90]:

```
excl = ["VWMe", "SMB", "HML", "MOM"]
for reg in excl:
    res = smf.ols(f"Ships ~ 1 + {reg}", data).fit(cov_type="HC0")
    pretty(f"{reg}: {res.pvalues[reg]}")
```

VWMe: 2.825662615113824e-87

SMB: 1.4899988412075129e-05

HML: 0.7930550353026412

MOM: 0.013617074267728941

In [91]:

```
excl.remove("VWMe")
for reg in excl:
    res = smf.ols(f"Ships ~ 1 + VWMe + {reg}", data).fit(cov_type="HC0")
    pretty(f"{reg}: {res.pvalues[reg]}")
```

SMB: 0.26284578560237704

HML: 7.813599736338563e-08

MOM: 0.385975338804352

In [92]:

```
excl.remove("HML")
for reg in excl:
    res = smf.ols(f"Ships ~ 1 + VWMe + HML + {reg}", data).fit(cov_type="HC0")
    pretty(f"{reg}: {res.pvalues[reg]}")
```

SMB: 0.057431312260540816

MOM: 0.9465305413853505

Model Selection

Information Criteria

- Information criteria trade-off fit and cost for additional penalties
- Two most common: AIC and BIC
- Select model that produces the smallest IC from candidate models

In [93]:

```
capm = smf.ols("Ships ~ 1 + VWMe", data).fit()
factor2 = smf.ols("Ships ~ 1 + VWMe + SMB", data).fit()
pretty(f"CAPM AIC: {capm.aic:0.1f}, BIC: {capm.bic:0.1f}")
pretty(f"2 Factor AIC: {factor2.aic:0.1f}, BIC: {factor2.bic:0.1f}")
```

CAPM AIC: 4225.9, BIC: 4234.9

2 Factor AIC: 4225.4, BIC: 4239.0

Model Selection

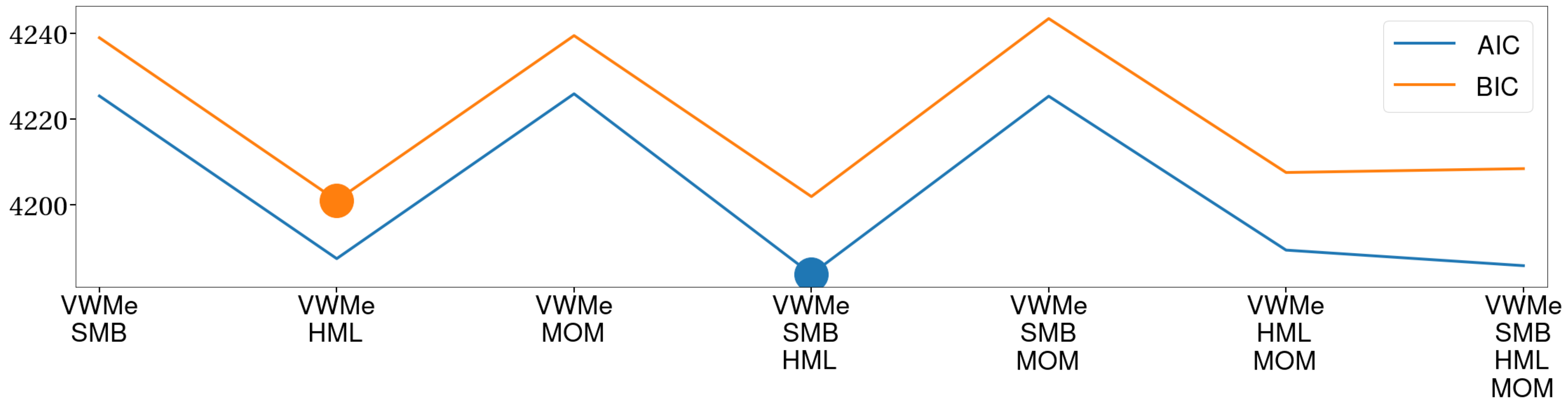
Global IC Search

In [126]:
```
pretty(ics.idxmin())
```

Out[126]:

AIC	VWMe,SMB,HML
BIC	VWMe,HML

In [127]:
```
plot_ics()
```



Model Selection

k-fold Cross-validation

- Focus on pseudo-out-of-sample prediction
- Split data into k equally sized random blocks
- Estimate parameters using $k - 1$ blocks
- Evaluate SSE using block not used in estimation
- Repeat k times in total computing the SSE once in each block
- Sum k SSE values into SSE_{xv}
- Choose model with the lowest out-of-sample SSE_{xv}

In [97]:

```
mod = "Ships ~ 1 + VWMe"
rg = np.random.default_rng(132217111120)
idx = rg.permutation(n)
fifth = n / 5
y = data.Ships.copy()
xv_errors = y.copy()

for i in range(5):
    reserve = idx[int(i * fifth) : int((i + 1) * fifth)]
    use = np.setdiff1d(idx, reserve)
    beta = smf.ols(mod, data.iloc[use]).fit().params
    xv_predictions = smf.ols(mod, data.iloc[reserve]).predict(beta)
    xv_errors.iloc[reserve] = y.iloc[reserve] - xv_predictions
sse_xv = (xv_errors ** 2).sum()
full_res = smf.ols(mod, data).fit()
pretty(f"XV SSE: {sse_xv:0.1f}, In-sample SSE: {n*full_res.mse_resid:0.1f}")
```

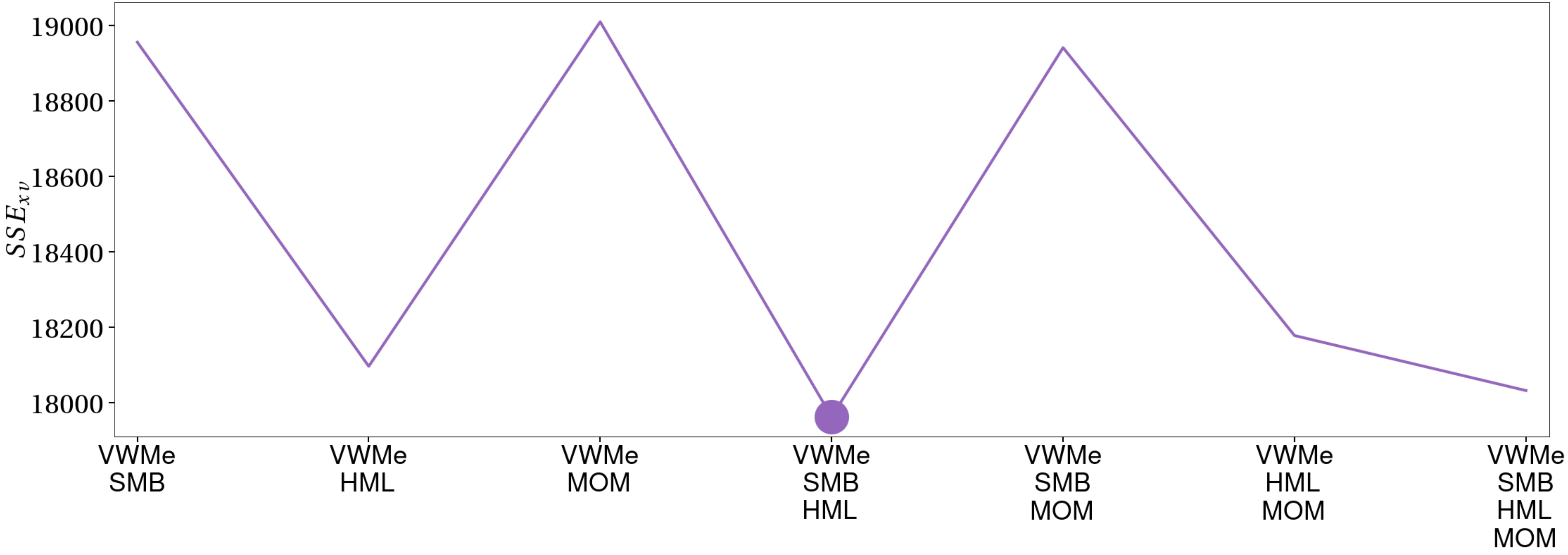
XV SSE: 19011.3, In-sample SSE: 18963.3

Model Selection

Cross-validation

In [133]:

```
xv_plot()
```



Analysis of Cross-Sectional Data

Checking for Specification Errors

- Testing Structural Stability
- Testing for Neglected Nonlinearities
- Visual Diagnostics
- Trimming and Windsorization

Specification Testing

The Chow Test

- Chow test is a stability test
- Implemented using dummy interaction variables

$$I_{[t>\tau]}$$

- Extend model with copy of variables interacted

$$Y_t = \mathbf{x}_t\boldsymbol{\beta} + I_{[t>\tau]}\mathbf{x}_t\boldsymbol{\gamma} + \epsilon_t$$

- Test using a Wald test (or LM or LR) with a χ^2_k distribution

In [100]:

```
mod = "Banks ~ 1 + VWMe + SMB + HML + MOM"
interact_mod = " + IxVWMe + IxSMB + IxHML + IxMOM"
ind = data.index > pd.to_datetime("1987-10-1")
ind
interact = data[["VWMe", "SMB", "HML", "MOM"]] * ind[:, None]
interact.columns = [f"Ix{col}" for col in interact]
both = pd.concat([data, interact], 1)
chow = smf.ols(mod + interact_mod, both).fit(cov_type="HC0")
summary(chow, [0, 1])
```

OLS Regression Results

Dep. Variable:	Banks	**R-squared:**	0.758
Model:	OLS	**Adj. R-squared:**	0.755

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.0588	0.126	-0.465	0.642	-0.307	0.189
VWMe	1.1137	0.057	19.469	0.000	1.002	1.226
SMB	-0.0774	0.072	-1.081	0.280	-0.218	0.063
HML	0.2557	0.086	2.990	0.003	0.088	0.423
MOM	-0.2023	0.059	-3.428	0.001	-0.318	-0.087
IxVWMe	0.0683	0.073	0.942	0.346	-0.074	0.211
IxSMB	-0.0842	0.089	-0.942	0.346	-0.259	0.091
IxHML	0.5119	0.103	4.947	0.000	0.309	0.715
IxMOM	0.0977	0.069	1.422	0.155	-0.037	0.232

In [101]:

```python
R = np.c_[np.zeros((4, 5)), np.eye(4)]
R
```

Out[101]:

```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

In [101]:

```python
R = np.c_[np.zeros((4, 5)), np.eye(4)]
R
```

Out[101]:

```
array([[0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

In [102]:

```python
chow_test = chow.wald_test(R)
chow_stat = float(chow_test.statistic)
chow_pvalue = chow_test.pvalue
pretty(f"The Chow statistic is {chow_stat:0.1f} and its p-value is {chow_pvalue:0.3f}")
```

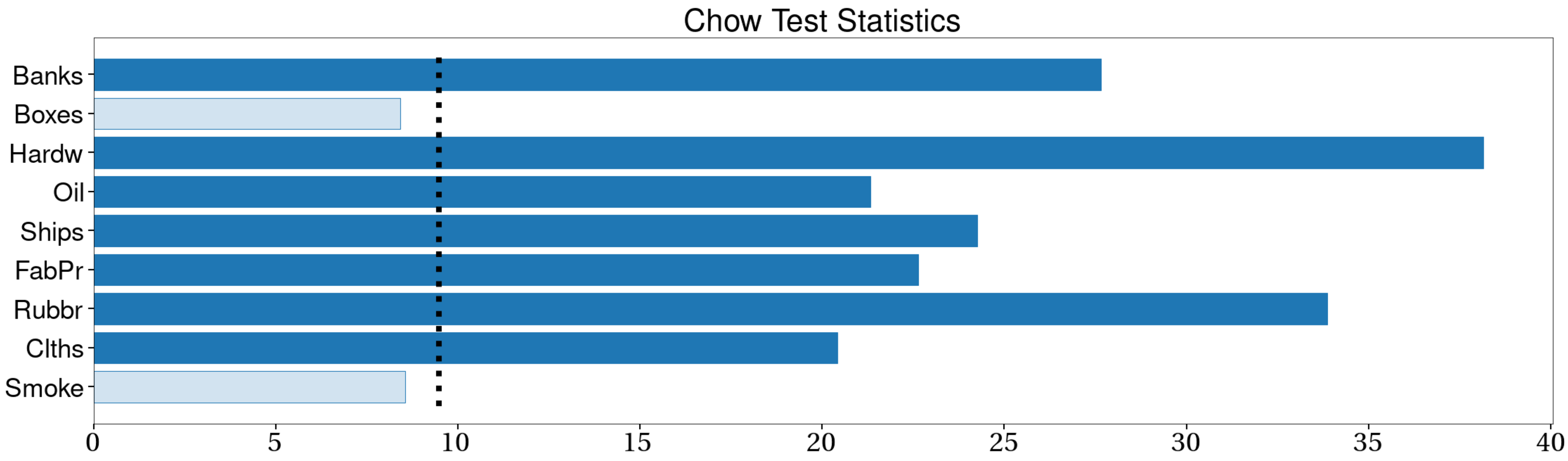
The Chow statistic is 27.7 and its p-value is 0.000

Chow Test

Industry Portfolios

In [104]:

```
cv = stats.chi2(4).ppf(0.95)
test_plot(chows, title="Chow Test Statistics", cv=cv)
_ = plt.plot([cv, cv], [-0.5, 8.5], "k:", linewidth=8)
```



Specification Testing

The RESET Test

- Test for general neglected nonlinearity
- Include powers of fitted value $\hat{Y}_i^p$ in the model
- Requires initial regression to generate fitted value $Y_i = \mathbf{x}_i\boldsymbol{\beta} + \gamma_2\hat{Y}_i^2 \left[+\gamma_3\hat{Y}_i^3\right] + \epsilon_i$

In [134]:

```
first_stage = smf.ols("FabPr ~ 1 + VWMe + SMB + HML + MOM", data).fit()
data["FabPrHat"] = first_stage.predict()
reset_res = smf.ols("FabPr ~ 1 + VWMe + SMB + HML + MOM + I(FabPrHat**2)", data).fit()
test_and_pval = pd.concat([reset_res.tvalues, reset_res.pvalues], 1)
test_and_pval.columns = ["t-stat", "p-value"]
test_and_pval
```

Out[134]:

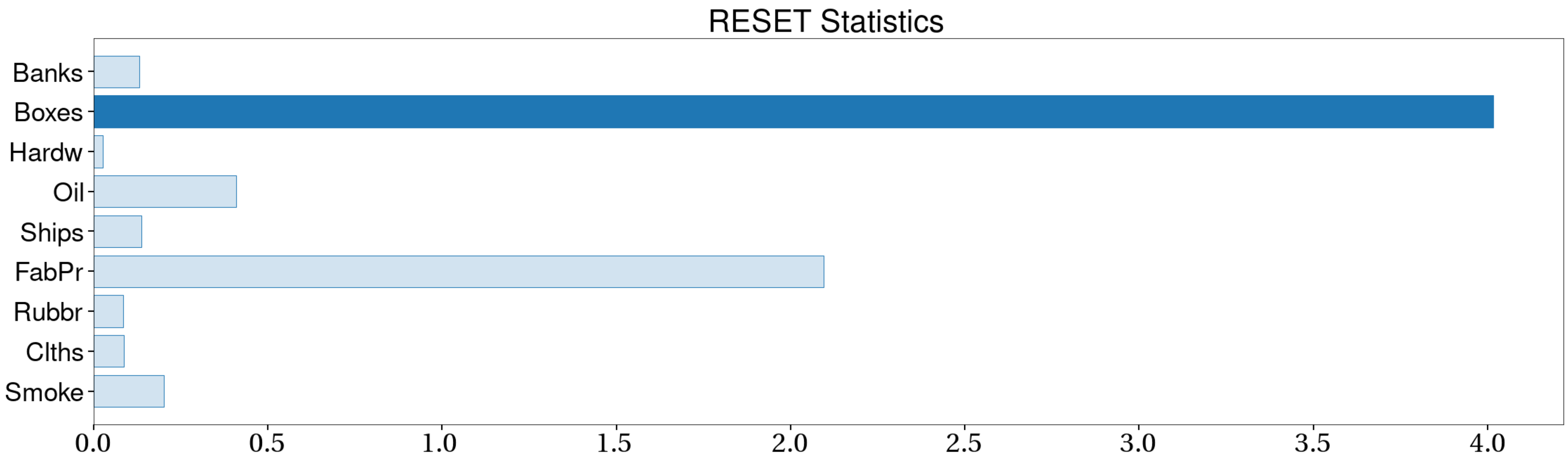
	t-stat	p-value
Intercept	-0.406132	6.847736e-01
VWMe	20.341804	3.244811e-72
SMB	8.877870	6.004409e-18
HML	3.095003	2.048885e-03
MOM	-2.513712	1.217672e-02
I(FabPrHat ** 2)	-1.953090	5.121869e-02

The RESET Test

Industry Portfolios

In [107]:

```
cv = stats.chi2(1).ppf(0.95)
test_plot(reset_stats, title="RESET Statistics", cv=cv)
```



Specification Testing

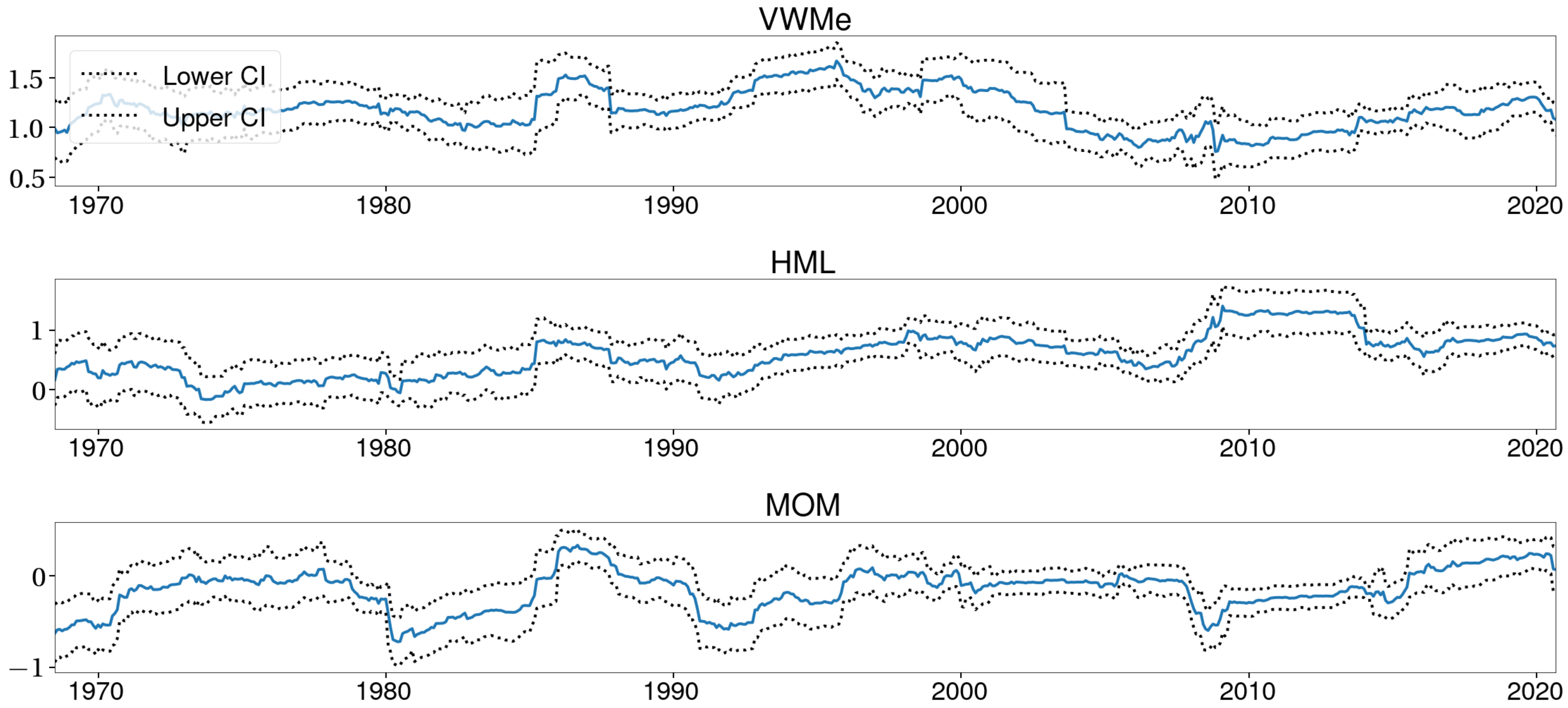
Rolling Regression Plots

- Estimate many regressions using a fixed window length
- Visual diagnostics for parameter stability
- Confidence intervals are approximate

In [109]:

```
from statsmodels.regression.rolling import RollingOLS

rolling_res = RollingOLS.from_formula(
    "Banks ~ 1 + VWMe + SMB + HML + MOM", data, window=60
).fit(cov_type="HC0")
fig = rolling_res.plot_recursive_coefficient(["VWMe", "HML", "MOM"])
```


VWMe
Lower CI
Upper CI
1.5
1.0
0.5
1970
1980
1990
2000
2010
2020
HML
1
0
1970
1980
1990
2000
2010
2020
MOM
0
−1
1970
1980
1990
2000
2010
2020

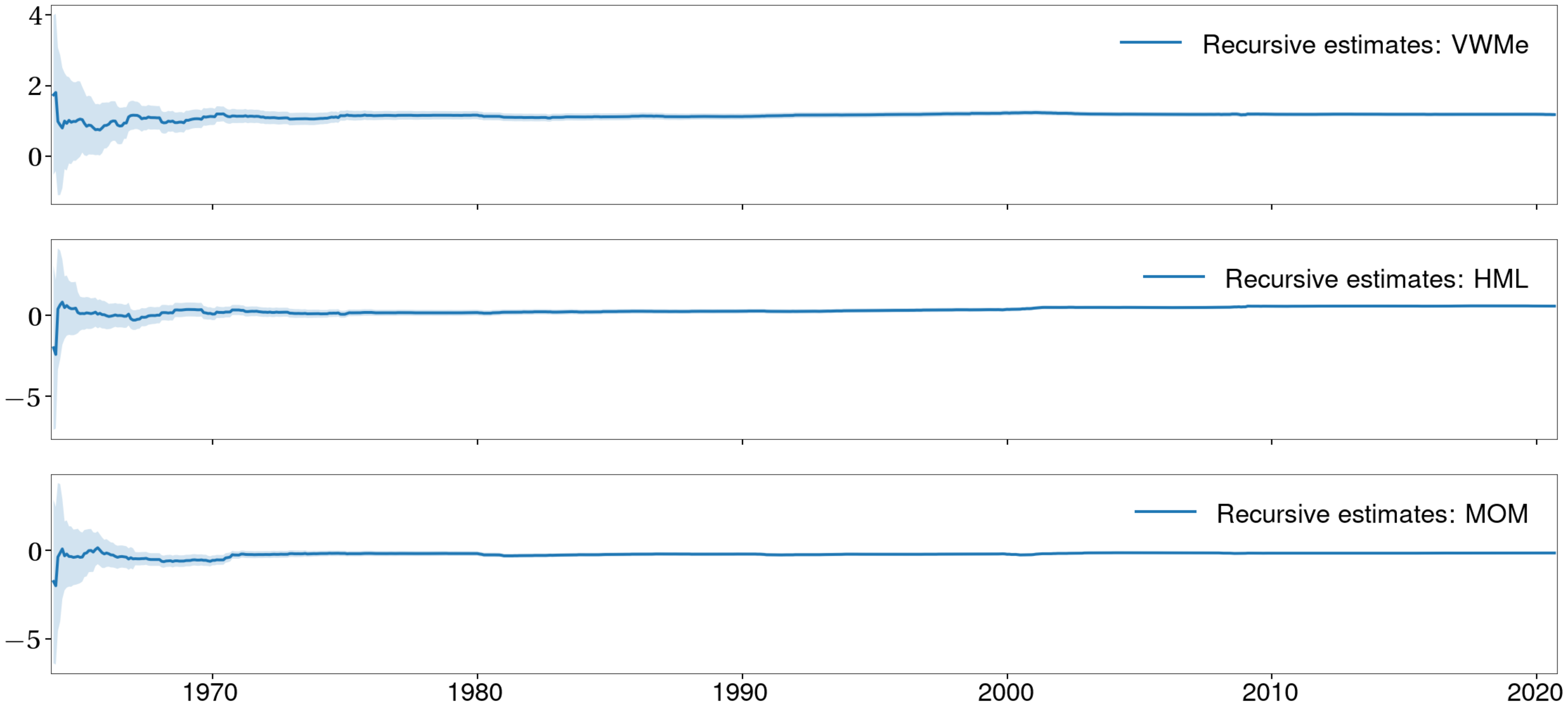
Specification Testing

Recursive Regression Plots

- Estimate many regressions using an expanding sample
- Visual diagnostics for parameter stability
- Confidence intervals are standard

In [111]:

```
recursive_plot()
```



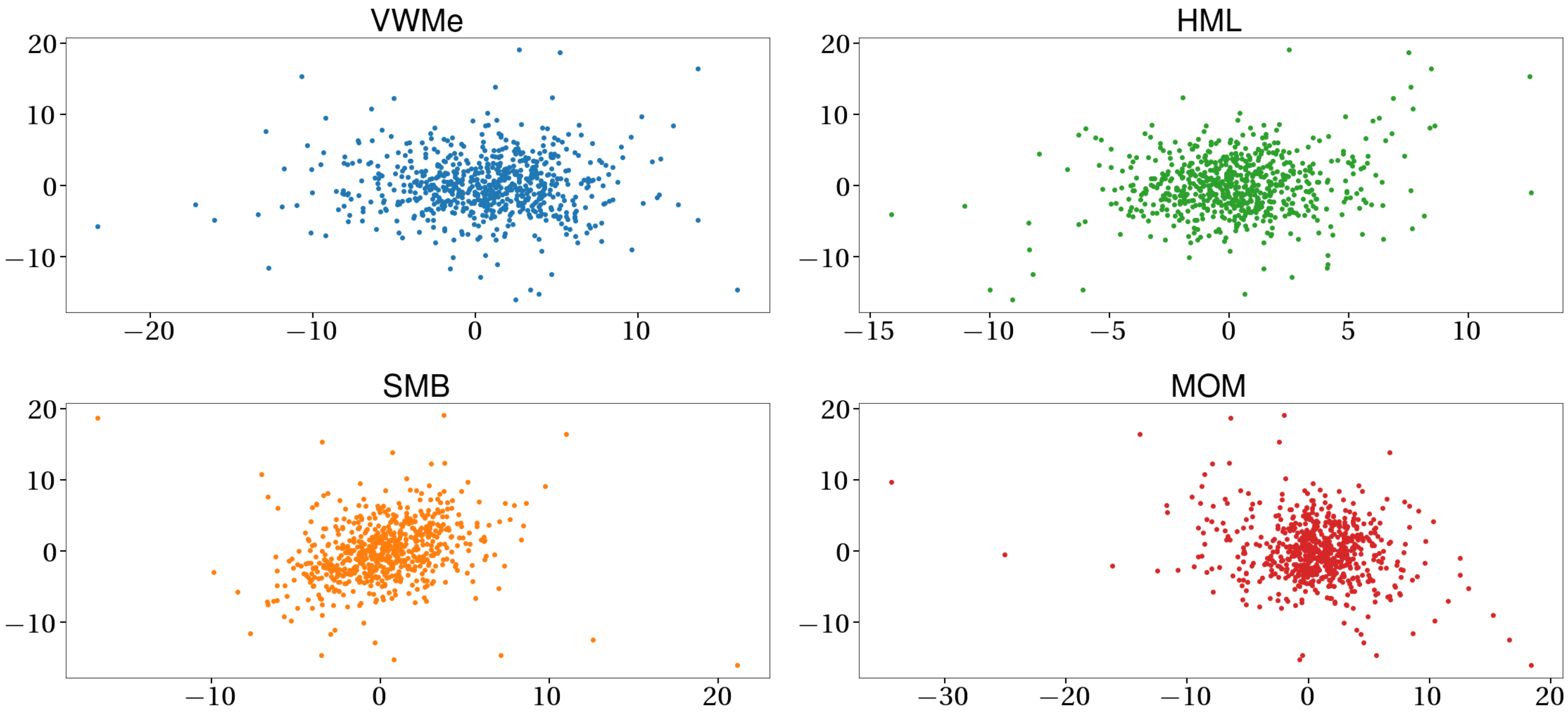
Specification Testing

Residual Plots: Residual vs X

- Residual plots are simple method to detect visible misspecification

In [113]:

```
plot_residual()
```



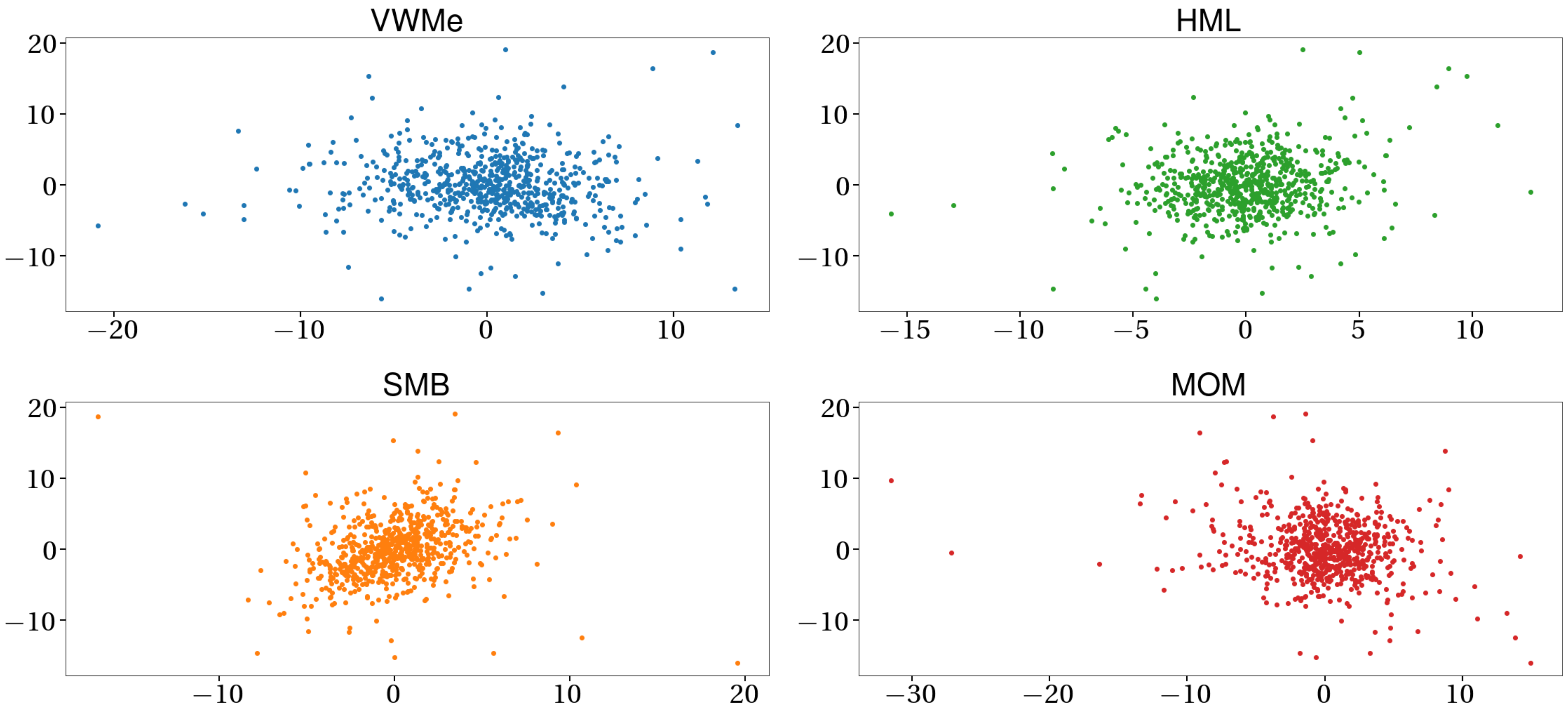
Specification Testing

Residual Plots: Residual vs $X_i | X_1, X_2, ..., X_{i-1}, X_{i+1}, ..., X_k$

- Sometimes useful to plot residuals against regressors *partialed* out
- Regress each regressor on the others included in the model
- Residual contains unique component of each regressor

In [115]:

```
plot_partial_residual()
```



Trimming

- Drop observations with large ϵ_i
- Requires an initial estimator of β
    - "Good" subset of data
    - Typical value in random subset
    - Robust estimator or LAD
- Remove observations with $\hat{\epsilon}_i$ below quantile α and above $1-\alpha$ for small α (1%, 2.5%, 5%)

In [116]:

```
lad = smf.quantreg("Oil ~ 1 + VWMe + SMB + HML + MOM", data).fit(q=0.5)
bounds = lad.resid.quantile([0.025, 0.975])
pretty(bounds)
```

Out[116]:

0.025	-7.929285
0.975	8.650551

In [117]:

```
retain = (lad.resid > bounds.iloc[0]) & (lad.resid < bounds.iloc[1])
trimmed = smf.ols("Oil ~ 1 + VWMe + SMB + HML + MOM", data.loc[retain]).fit(
    cov_type="HC0"
)
summary(trimmed)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.1302	0.141	-0.925	0.355	-0.406	0.146
VWMe	0.9070	0.038	23.716	0.000	0.832	0.982
SMB	-0.1617	0.055	-2.918	0.004	-0.270	-0.053
HML	0.3853	0.062	6.224	0.000	0.264	0.507
MOM	0.0567	0.039	1.448	0.148	-0.020	0.133

In [118]:

```
full = smf.ols("Oil ~ 1 + VWMe + SMB + HML + MOM", data).fit(cov_type="HC0")
summary(full)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.1240	0.172	-0.722	0.470	-0.461	0.213
VWMe	0.9537	0.053	17.951	0.000	0.850	1.058
SMB	-0.1525	0.065	-2.335	0.020	-0.281	-0.024
HML	0.3944	0.089	4.414	0.000	0.219	0.569
MOM	0.0609	0.050	1.218	0.223	-0.037	0.159

Windsorization

- Similar to trimming with one key difference
- Replace large ϵ_i with their quantile
- No data dropped

In [119]:

```
data["Windsorized"] = data.Oil
predicted = lad.predict()
low = lad.resid < bounds.iloc[0]
data.loc[low, "Windsorized"] = predicted[low] + bounds.iloc[0]
high = lad.resid > bounds.iloc[1]
data.loc[high, "Windsorized"] = predicted[high] + bounds.iloc[1]
windsorized = smf.ols("Windsorized ~ 1 + VWMe + SMB + HML + MOM", data).fit(
    cov_type="HC0"
)
summary(windsorized)
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.1163	0.154	-0.755	0.451	-0.419	0.186
VWMe	0.9225	0.042	21.844	0.000	0.840	1.005
SMB	-0.1519	0.059	-2.568	0.010	-0.268	-0.036
HML	0.3700	0.069	5.389	0.000	0.235	0.505
MOM	0.0564	0.043	1.303	0.193	-0.028	0.141